



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**SOFTWARE COMMUNICATIONS ARCHITECTURE
(SCA) COMPLIANT SOFTWARE DEFINED RADIO
DESIGN FOR IEEE 802.16 WIRELESSMAN-OFDM™
TRANSCIVER**

by

Kian Wai, Low

December 2006

Thesis Advisor:
Second Reader:

Frank Kragh
Clark Robertson

Approved for public release, distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2006	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Software Communications Architecture (SCA) Compliant Software Defined Radio Design for IEEE 802.16 Wirelessman-OFDM™ Transceiver			5. FUNDING NUMBERS	
6. AUTHOR(S) Low Kian Wai				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) <p>Demands for seamless mobile communications are driving the research and development of software defined radio (SDR), which enables a single terminal to transmit and receive in distinct wireless systems through a simple change in software to reconfigure the terminal's functions. Its application areas include military use, home networks, intelligent transport systems and cellular communications. Several SDR software architectures have been developed during the last few years. One implementation of the Software Communications Architecture is the Open Source SCA Implementation::Embedded (OSSIE) which is developed by the Mobile and Portable Radio Research Group (MPRG) at Virginia Tech. The goal of this thesis was to design and implement transmitter and receiver components using OSSIE. The components were designed for use in the IEEE 802.16 WirelessMAN-OFDM™ transceiver and for contribution to the library of components being developed. Thus, the components will be flexible and useful for other transceivers by specifying the appropriate parameters.</p>				
14. SUBJECT TERMS Software Defined Radio, OSSIE, Software Communications Architecture, IEEE 802.16.			15. NUMBER OF PAGES 98	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release, distribution is unlimited

**SOFTWARE COMMUNICATIONS ARCHITECTURE (SCA) COMPLIANT
SOFTWARE DEFINED RADIO DESIGN FOR IEEE 802.16 WIRELESSMAN-
OFDMTM TRANSCEIVER**

Kian Wai, Low
Major, Republic of Singapore Air Force
B.Eng (EE) (Hons)., Nanyang Technological University, 1998

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
December 2006**

Author: Kian Wai, Low

Approved by: Frank Kragh
Thesis Advisor

Clark Robertson
Second Reader

Jeffrey B. Knorr
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Demands for seamless mobile communications are driving the research and development of software defined radio (SDR), which enables a single terminal to transmit and receive in distinct wireless systems through a simple change in software to reconfigure the terminal's functions. Its application areas include military use, home networks, intelligent transport systems and cellular communications. Several SDR software architectures have been developed during the last few years. One implementation of the Software Communications Architecture is the Open Source SCA Implementation::Embedded (OSSIE) which is developed by the Mobile and Portable Radio Research Group (MPRG) at Virginia Tech. The goal of this thesis was to design and implement software defined radio transmitter and receiver components using OSSIE. The components were designed for use in the IEEE 802.16 WirelessMAN-OFDM™ transceiver and for contribution to the library of components being developed. Thus, the components will be flexible and useful for other transceivers by specifying the appropriate parameters.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	INTRODUCTION.....	1
B.	THESIS OBJECTIVE.....	1
C.	THESIS OUTLINE.....	2
II.	BACKGROUND	3
A.	SOFTWARE DEFINED RADIO	3
1.	What Is a Software Defined Radio?	3
2.	Benefits of a Software Defined Radio.....	4
B.	IEEE 802.16 WIRELESSMAN™ Standard	5
1.	Broadband Wireless Access	5
a.	<i>Benefits</i>	5
b.	<i>Architecture</i>	6
2.	Overview of IEEE 802.16 WirelessMAN™ Standard.....	7
a.	<i>General Specifications</i>	7
b.	<i>Project Development Milestones</i>	8
3.	IEEE 802.16 WirelessMAN-OFDM™ Physical Layer	8
a.	<i>OFDM Symbol Description</i>	8
b.	<i>Channel Coding</i>	9
c.	<i>Randomization</i>	9
d.	<i>Forward Error Correction</i>	10
e.	<i>Interleaving</i>	12
f.	<i>Modulation</i>	13
g.	<i>Pilot Modulation</i>	14
C.	SUMMARY	5
III.	SOFTWARE DESIGN ENVIRONMENT	17
A.	SOFTWARE ARCHITECTURE.....	17
1.	Software Communications Architecture (SCA).....	17
2.	Common Object Request Broker Architecture (CORBA)	18
B.	OPEN SOURCE SCA IMPLEMENTATION::EMBEDDED (OSSIE) ...	19
C.	WAVEFORM DEVELOPMENT.....	19
D.	OSSIE WAVEFORM DEVELOPER (OWD)	20
1.	File and Directory Structure.....	20
2.	XML Domain Profile	21
3.	C++ Code	22
E.	SUMMARY	5
IV.	SOFTWARE DEVELOPMENT	25
A.	APPROACH.....	25
1.	Incremental Development Model	25
B.	CONSIDERATIONS	27
1.	Assumptions	27
2.	Functionality.....	28
3.	Reusability and Reconfigurability.....	29

	4. Constraints.....	29
C.	WAVEFORM DESIGN.....	30
D.	COMPONENT DESIGN.....	32
	1. Transmitter.....	33
	a. Randomizer.....	33
	b. Convolutional Encoder.....	34
	c. Interleaver.....	35
	d. BPSK Symbol Mapper.....	35
	e. QPSK Symbol Mapper.....	36
	f. 16-QAM Symbol Mapper.....	36
	g. 64-QAM Symbol Mapper.....	37
	h. Insert Guard Subcarriers.....	37
	i. Insert Pilot Tone and DC Null Subcarriers.....	38
	j. Inverse Fast Fourier Transform (IFFT).....	38
	k. Insert Cyclic Prefix.....	39
	2. Receiver.....	40
	a. De-randomizer.....	40
	b. Convolutional Decoder.....	41
	c. De-interleaver.....	43
	d. BPSK Symbol De-mapper.....	44
	e. QPSK Symbol De-mapper.....	44
	f. 16-QAM Symbol De-mapper.....	45
	g. 64-QAM Symbol De-mapper.....	45
	h. Remove Guard Subcarriers.....	45
	i. Remove Pilot Tone and DC Null Subcarriers.....	46
	j. Fast Fourier Transform (IFFT).....	47
	k. Remove Cyclic Prefix.....	48
E.	SUMMARY.....	5
V.	SOFTWARE TESTS AND RESULTS.....	51
	A. TEST METHODOLOGY.....	51
	B. RESULTS.....	54
	1. Functionality.....	54
	2. Reusability and Reconfigurability.....	55
	C. SUMMARY.....	5
VI.	ADDITIONAL WORK.....	61
	A. MULTI-MODE OPERATIONS WAVEFORM DESIGN.....	61
	B. MULTI-MODE OPERATIONS COMPONENT DESIGN.....	63
	C. TESTS AND RESULTS.....	65
	D. SUMMARY.....	5
VII.	CONCLUSIONS AND RECOMMENDATIONS.....	69
	A. CONCLUSIONS.....	69
	B. RECOMMENDATIONS FOR FUTURE WORK.....	69
	LIST OF REFERENCES.....	71
	INITIAL DISTRIBUTION LIST.....	73

LIST OF FIGURES

Figure 1.	The ideal software radio architecture.....	3
Figure 2.	A typical practical software radio architecture.	4
Figure 3.	Fixed broadband wireless access (From [11]).	6
Figure 4.	OFDM symbol time structure (From [10]).	8
Figure 5.	OFDM frequency description (From [10]).	9
Figure 6.	PRBS generator for data randomization (From [10]).	10
Figure 7.	Convolutional encoder of rate 1/2 (From [10]).	11
Figure 8.	BPSK, QPSK, 16-QAM and 64-QAM constellations (From [10]).	14
Figure 9.	PRBS for pilot modulation (From [10]).	15
Figure 10.	SCA Core Framework IDL relationships (From [17]).	18
Figure 11.	OWD generated directory layout (From [20]).	20
Figure 12.	Incremental Development Model (After [22]).	26
Figure 13.	Assumed SDR architecture design.....	28
Figure 14.	Conceptual waveform design.....	31
Figure 15.	Working transmitter waveform design.	31
Figure 16.	Working receiver waveform design.....	31
Figure 17.	Functional architecture of the OWD auto-generated C++ code.	32
Figure 18.	Functional description of component <i>randomizer_802_16</i>	34
Figure 19.	Functional description of component <i>CC_encoder_802_16</i>	35
Figure 20.	Functional description of component <i>interleaver_802_16</i>	35
Figure 21.	Functional description of component <i>bpsk_mod_802_16</i>	36
Figure 22.	Functional description of component <i>guardIns_802_16</i>	37
Figure 23.	Functional description of component <i>ptIns_802_16</i>	38
Figure 24.	Functional description of component <i>Data_IFFT</i> (After [22]).	39
Figure 25.	Functional description of component <i>cpIns_802_16</i>	40
Figure 26.	Functional description of component <i>derandomizer_802_16</i>	41
Figure 27.	Functional description of component <i>Data_conv_dec</i> (After [22]).	42
Figure 28.	Functional description of function <i>initialize_viterbi</i> within Component <i>Data_conv_dec</i> (From [22]).	42
Figure 29.	Functional description of function <i>process_viterbi</i> within component <i>Data_conv_dec</i> (From [22]).	43
Figure 30.	Functional description of component <i>deinterleaver_802_16</i>	43
Figure 31.	Functional description of component <i>bpsk_demod_802_16</i>	44
Figure 32.	Functional description of component <i>qpsk_demod_802_16</i>	45
Figure 33.	Functional description of component <i>guardRem_802_16</i>	46
Figure 34.	Functional description of component <i>ptRem_802_16</i>	47
Figure 35.	Functional description of component <i>Data_FFT</i> (After [22]).	48
Figure 36.	Functional description of component <i>cpRem_802_16</i>	48
Figure 37.	Software test methodology.	52
Figure 38.	Set-up for testing a transmitter component.	52
Figure 39.	An example of set-up for testing a receiver component.	53
Figure 40.	An example of set-up for testing an incremented waveform.	53

Figure 41.	An example of a SDR component description file.	56
Figure 42.	An example of data handling within a component.	58
Figure 43.	Conceptual waveform design for multi-mode operations.....	62
Figure 44.	Working model of a multi-mode operations waveform.....	63
Figure 45.	Functional Description of <i>SelectorReal_3Out</i>	64
Figure 46.	Script of test results for a multi-mode operations capable waveform.	66

LIST OF TABLES

Table 1.	Air interface nomenclature (From [10]).	7
Table 2.	OFDM symbol parameters (From [10]).	9
Table 3.	The inner convolutional code with puncturing configuration (From [10]).	11
Table 4.	Mandatory channel coding per modulation (From [10]).	12
Table 5.	SCA development environment comparison (From [20]).	20
Table 6.	OWD generated waveform files (From [20]).	21
Table 7.	OWD generated component files (From [20]).	21
Table 8.	An example of an IEEE 802.16 test case (From [10]).	54

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to give thanks to my Lord and Saviour, Jesus Christ for His grace and blessings that made all this possible. I must thank my wonderful wife, Tricia, who has supported and taken good care of me while I toil away in this thesis. I would like to thank my thesis advisor, Assistant Professor Frank Kragh for the thesis opportunity, and his kind understanding and patience with me. I would also like to thank Professor Clark Robertson for his good instruction on the necessary knowledge fields that helped prepare me for this thesis. Finally, I am also grateful to my organization, the Republic of Singapore Air Force, for giving me this opportunity to study at the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Demands for seamless mobile communications are driving the research and development of software defined radio (SDR), which enables a single terminal to transmit and receive in distinct wireless systems through a simple change in software to reconfigure the terminal's functions. Its application areas include military use, home networks, intelligent transport systems and cellular communications.

The SDR technology allows radio systems to be dynamically reprogrammed to support new air interface standards or to provide new features and capabilities to the radio while in service. The benefit of SDR lies in its ability to support interoperation of a single radio device on multiple radio networks. The flexibility and reconfigurability of the SDR enables the radio architecture to support new waveform standards as they emerge.

The Software Communications Architecture (SCA) is an open architecture framework that specifies the structure and operations within a SDR. It is a requirements specification for the design of the SDR. Despite its origins in the military domain, the SCA has also been widely accepted in commercial applications [5][6]. Several SDR software architectures have been developed during the last few years. One implementation of the SCA is the Open Source SCA Implementation::Embedded (OSSIE) which is developed by the Mobile and Portable Radio Research Group (MPRG) at Virginia Tech [7].

OSSIE is a C++-based open source implementation of the SCA. Still a beta version release, the software also comes with a tool called the OSSIE Waveform Developer (OWD) for the rapid development of the SDR components and application waveforms. An evolving library of SDR components is also available on the MPRG's website. [7]

The goal of this thesis is to design and implement software defined radio transmitter and receiver components using OSSIE. The components will be designed for use in an IEEE 802.16 WirelessMAN-OFDMTM transceiver and for contribution to the library of components being developed. Thus the components will be flexible and useful

for other transceivers by specifying the appropriate parameters. For this thesis, the SDR waveforms and components were built using OSSIE version 0.5.0 that implements version 2.2.1 of the SCA standard.

This research was the first attempt to use OSSIE to implement a SDR transceiver for the IEEE 802.16 standard so the software radio components were designed from scratch . Thus, the Incremental Development Model was adopted as a software process model to structure the development of the software components. The intent is to develop the application waveform, which comprises of SDR components, incrementally and systematically [21]. The process starts with a simple implementation of a subset of the software requirements, in this case the SDR components, and iteratively enhances the evolving versions until the full system, i.e. the SDR application waveform, is implemented. The incremental development model consists of three stages: Design, Develop and Verify [22].

Several considerations drive the design of the SDR components. These considerations are translated from the objectives of this thesis. These considerations affect the decisions that drive the subsequent design of the SDR components. They are:

- Assumptions on the hardware platform that runs the software as well as provide the RF front-end of the system. The software design assumes that the incoming signal to the waveform consist of frequency down-converted to the complex baseband, discrete samples.
- Functionality of the IEEE 802.16 WirelessMAN-OFDMTM standard, which is applicable to licensed bands from 2 to 11GHz, upon which this thesis will concentrate. This refers to a basic single-mode configuration of the physical layer.
- Reusability and reconfigurability of the software components which should be useful in the library for building other waveforms with minimal or no amendments needed. This requires the components to be simple and elementary such that they are single-functioned and have generic port interfaces.

- Constraints of OSSIE 0.5.0 being a beta version at the time of this thesis work where a single port interface of a component is not allowed to have multiple connections to other components. Also, a common buffer for each component is used for data transactions through multiple input port interfaces of the same type. There is an additional constraint to the testing of the developed SDR components given that the interface to the hardware RF front-end will not be implemented in this thesis.

The transmitter and receiver waveform were broken down into simple and elementary components listed in the following as associated pairs:

- Randomizer and de-randomizer
- Convolutional encoder and Viterbi decoder
- Interleaver and de-interleaver
- BPSK symbol mapper and symbol de-mapper
- QPSK symbol mapper and symbol de-mapper
- 16-QAM symbol mapper and symbol de-mapper
- 64-QAM symbol mapper and symbol de-mapper
- Insert and remove guard subcarriers
- Insert and remove pilot tone and DC null subcarriers
- Fast Fourier Transform and Inverse Fast Fourier Transform
- Insert and remove Cyclic Prefix

A test case is available in the IEEE 802.16 standard document to validate the functionality of the individual components. The SDR transmitter components were first tested individually with the results verified against the test case to validate their functionality. Having successfully tested and validated the transmitter component, the corresponding receiver component would then be tested against this transmitter component. Following the successful tests of the components, they would then be integrated into the waveform as increments and be tested as a waveform.

All basic transmitter and receiver components required to build the waveform compliant to the physical layer of the IEEE 802.16 WirelessMAN-OFDM, except the Reed-Solomon encoder and decoder, were tested and had their functionality validated successfully. This does not include functions of phase synchronization, fading mitigation, and multiple antennae reception.

The SDR components developed in this thesis were designed with a strong emphasis on reusability and reconfigurability. The following describes the features incorporated into the design of the components that strives to achieve these traits:

- Documentation with proper commentary in the software source code and a description file for each component that details the general parameters relevant to understanding the component's structural design.
- Proper naming convention of each component that will allow in the library, where applicable, multiple components of the same function but different attributes. This alleviates the need to change parameters, which incurs recompilation of the component, to accommodate different waveforms.
- Dynamic data size handling of each component enabling them to accommodate waveforms with different data size requirements.
- Elementary and single-function design that enables the components to serve as basic building blocks in building any waveform.
- Generic port interface configuration design where only data port interfaces were provisioned for that will alleviate complexity in waveform design.

As additional design work in this thesis, multimode operations were explored using a waveform design which uses simple and reusable components having the traits above. The waveform consists of multiple subsets that encompass different modes of operations. A selector that precedes these waveform subsets receives control information and routes the received signal to the appropriate waveform subset. A receiver at the end of these waveform subsets receive the processed data stream and extracts control information from the header and feeds it back to the selector.

The main purpose of the experiment was to test the synchronization of the signal flow among the components which was critical to rendering the design feasible for multi-mode operations. The waveform was constructed in OSSIE Waveform Developer (OWD) and tested successfully demonstrating that multi-mode operations within a waveform is feasible without compromising the reusability of the components. The waveform can be reconfigured with different profiles easily.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF SYMBOLS, ACRONYMS, AND/OR ABBREVIATIONS

ADC	Analog to Digital Converter
ADSL	Asynchronous Digital Subscriber Line
ASIC	Application Specific Integrated Circuit
BWA	Broadband Wireless Access
BS	Base Station
CORBA	Common Object Request Broker Architecture
COTS	Commercially Off-The-Shelf
CF	Core Framework
CP	Cyclic Prefix
DAC	Digital to Analog Converter
DL	Downlink
DSP	Digital signal processor
FDD	Frequency Division Duplexing
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FPGA	Field programmable gate-arrays
IDL	Interface Definition Language
IF	Intermediate Frequency
IFFT	Inverse Fast Fourier Transform
JTRS	Joint Tactical Radio System
LNA	Low Noise Amplifier
LO	Local Oscillator
MAC	Medium Access Control
MP-MP	Multipoint-to-Multipoint
NLOS	No Line-Of-Sight
OE	Operating Environment
OMG	Object Management Group
ORB	Object Request Broker
OSSIE	Open Source SCA Implementation::Embedded
OWD	OSSIE Waveform Developer
PMP	Point-to-Multipoint
PRBS	Pseudo Random Binary Sequence
RF	Radio Frequency
RS	Repeater Station
SCA	Software Communications Architecture
SDR	Software Defined Radio
SS	Subscriber Station
TDD	Time Division Duplexing
TE	Terminal Equipment
UL	Uplink
UML	Unified Modeling Language
WDE	Waveform Development Environment
XML	Extensible Markup Language

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. INTRODUCTION

Software defined radio (SDR) is basically a radio implemented in software. Ideally, the communication signal is all processed in software within an SDR. This enables a reconfigurable system architecture for wireless networks and user terminals. A strong motivation for SDR technology is that it enables building of multi-mode, multi-band and multi-functional wireless devices that can be improved simply via software upgrades without need to recall hardware units [1].

One of the first software defined radio architectures was the SPEAKeasy system, initiated in the early 1990's by the US Air Force and eventually turning into a joint effort by the US military branches [3]. However, an important milestone in the proliferation of SDR is the US Department of Defense Joint Tactical Radio System (JTRS) project started in 1990's to develop a common programmable and multi-functional radio system that can be used for communication among all the Services [4]. The JTRS program developed an open Software Communications Architecture (SCA) which is an open architecture framework that specifies the operations within a SDR. Despite its origins in the military domain, the SCA has been widely accepted in commercial applications [5][6].

Open Source SCA Implementation::Embedded (OSSIE) is a C++-based open source implementation of the SCA. Still a beta version release, the software also comes with a tool called the OSSIE Waveform Developer (OWD) for the rapid development of the SDR components and application waveforms. An evolving library of SDR components is also available on the Mobile and Portable Radio Research Group's (MPRG) website. [7]

B. THESIS OBJECTIVE

The objective of this thesis is to develop components of a software defined radio receiver using OSSIE. The components shall be compliant to the physical layer of the IEEE 802.16 WirelessMan-OFDMTM standard. The components shall also contribute to

the library that will serve as a repository from which components can be retrieved to build other transceivers. Thus, the components shall be designed such that they are easily reusable and reconfigurable.

C. THESIS OUTLINE

The organization of the thesis is as follows: Chapter II elaborates on the Software Defined Radio as a system, its benefits and challenges in implementation. The chapter also introduces the IEEE 802.16 WirelessMANTM standard as a concept before elaborating on the physical layer of the IEEE 802.16 WirelessMAN-OFDMTM. Chapter III elaborates on the design environment for the development of the transceiver components in software. Chapter IV elaborates on the approach and considerations in the software design of the transceiver components. The chapter also presents the design algorithm of the transceiver components. Chapter V elaborates the methodology for test and verification of the transceiver components. The chapter then presents the test and verification results with respect to the component functionality. The chapter also presents the design features with respect to ensuring component reusability. Lastly, Chapter VI concludes the thesis and highlights future work.

II. BACKGROUND

A. SOFTWARE DEFINED RADIO

1. What is a Software Defined Radio?

The term software radio, used interchangeably with software defined radio, is basically a radio implemented in software. In other words, the communication signal is sampled and processed digitally within a radio. For the receiver, the hardware consists firstly of an antenna system that receives wideband communication signals. There would also be an analog-to-digital converter (ADC) that samples and digitizes the signal. Processing of the signal from here on would be done in software. The software can be loaded on any general purpose processors, field programmable gate-arrays (FPGA), digital signal processors (DSP) and application-specific integrated circuit (ASIC). [8]

Joe Mitola stated that a true software radio places the software, “as close to the antenna as possible” [9]. In other words, an ideal software radio is one where analogue to digital conversion takes place immediately after the antennae and all subsequent processing is carried out in software. Figure 1 shows an ideal software radio where two antennae are shown. All the main functions are carried out in software including the RF and IF processing of the signals, followed by the baseband functions such as modulation and demodulation. The disadvantage of this architecture is that the entire RF spectrum is converted by the ADC. This imposes very high performance demands on the ADC in terms of bandwidth, dynamic range and sampling rate. Current signal conversion technology is not established to realize such an ideal SDR as yet.

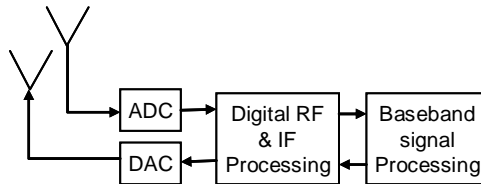


Figure 1. The ideal software radio architecture.

Conventionally, a practical software defined radio is implemented with analog RF front-end circuitry as shown in Figure 2. The analog signal is down-converted to IF before being digitized by the ADC for follow-on processing in software. The main challenge therefore in progressing towards the ideal software radio architecture lays in the realization of fast, wide-band, high-resolution and economical ADC and DAC.

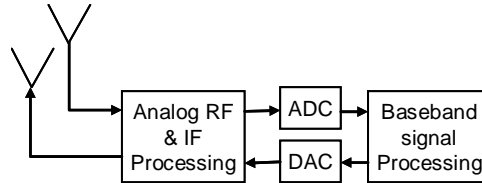


Figure 2. A typical practical software radio architecture.

2. Benefits of a Software Defined Radio

In his book, *Software Radio: A Modern Approach to Radio Engineering*, Dr. Jeffrey Reed summarized with five factors that are expected to push wider acceptance of software radio [2].

a. Multifunctionality

The same piece of hardware i.e. the radio set can be used to transmit, receive and process different communication signals that adhere to different air interface standards. This can be done simply by reconfiguring the software.

b. Global Mobility

The same piece of hardware i.e. the radio set can be used in different parts of the world that endorse different air interface standards. This can again be done simply by reconfiguring the software.

c. Compactness and Power Efficiency

Unlike traditional non-SDR systems which require multiple hardware sets for multi-functional communication, the same piece of SDR hardware can be reused for such a purpose. This results in a compact and power-efficient design, especially as the number of systems increases.

d. Ease of Manufacture

A SDR comprises of fewer hardware parts than a traditional radio since most processing is done in software within a general purpose microprocessors or special purpose microprocessors like the DSP, or in reconfigurable hardware including FPGAs. This eases the production cycle for the manufacturer with lesser parts to standardize and produce.

e. Ease of Upgrades

Any service upgrade can be easily introduced through the release of new software versions without the expense of recalling or replacing the hardware units. A user can simply download the software off the internet and load it into the SDR.

B. IEEE 802.16 WIRELESSMAN™ Standard

1. Broadband Wireless Access

Broadband Wireless Access (BWA) is a technology aimed at providing high-speed wireless access over a wide area from devices such as personal computers to data networks. According to the IEEE 802.16-2004 standard, broadband means having instantaneous bandwidth greater than around 1 MHz and supporting data rates greater than about 1.5 Mbit/s [10].

a. Benefits

BWA has become the best way to meet increasing demand for fast internet connection and integrated data, voice and video services. In addition to providing capacity that supports high data rates, BWA is wireless which enables a faster, more convenient and easier infrastructure set-up over the wired networks. For the users, BWA also means a possibility of mobile data communications. [12]

b. Architecture

Fixed BWA systems typically include at least a base station (BS) and a number of subscriber stations (SS). The BS connects the user SS to a core network. An uplink connection has the SS transmitting to the BS. The reverse is true for a downlink connection. Typically a BS uses several directional antennae and employs a sectoring technique to provide a 360 degree area coverage. Within a given frequency channel and BS antenna sector, all SS receive the same transmission. As such, the available bandwidth is shared among the SS users within the coverage area. This can be achieved through time-division multiple access or frequency-division multiple access. The shared bandwidth can be distributed via on-demand or fixed allocation methods. A reference fixed BWA system is shown in Figure 3. [11]

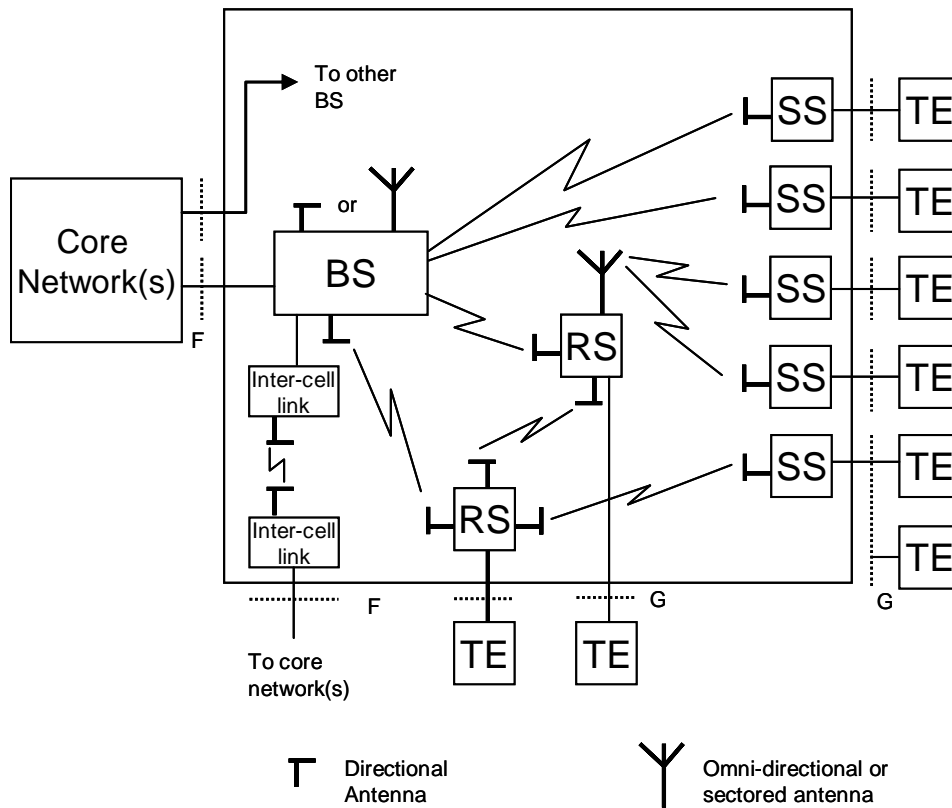


Figure 3. Fixed broadband wireless access (From [11]).

2. Overview of IEEE 802.16 WirelessMAN™ Standard

The Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA) sought to make BWA more widely available by developing IEEE Standard 802.16, which specifies the wireless metropolitan area network (WirelessMAN) Air Interface.

a. General Specifications

IEEE 802.16 focuses on the efficient use of bandwidth between 10 and 66 GHz and also the 2 to 11 GHz region. The standard defines a medium access control (MAC) layer that supports multiple physical layer specifications customized for the frequency band of use. The 10 to 66 GHz standard supports licensed frequencies for two-way Line-Of-Sight (LOS) communications. The 2 to 11 GHz standard supports both unlicensed and licensed bands without need for LOS communication. [12]

Table 1 summarizes the nomenclature for the various air interface specifications in this standard.

Designation	Applicability	PHY	Additional MAC requirements	Options	Duplexing alternative
WirelessMAN-SC™	10-66 GHz	8.1			TDD FDD
WirelessMAN-SCa™	Below 11 GHz licensed bands	8.2		AAS (6.3.7.6) ARQ (6.3.4) STC (8.2.1.5.3)	TDD FDD
WirelessMAN-OFDM™	Below 11 GHz licensed bands	8.3		AAS (6.3.7.6) ARQ (6.3.4) Mesh (6.3.6.6) STC (8.3.8)	TDD FDD
WirelessMAN-OFDMA	Below 11 GHz licensed bands	8.4		AAS (6.3.7.6) ARQ (6.3.4) STC (8.4.8)	TDD FDD
WirelessHUMAN™	Below 11 GHz licensed-exempt bands	[8.2, 8.3, or 8.4] and 8.5	DFS (6.3.15)	AAS (6.3.7.6) ARQ (6.3.4) Mesh (6.3.6.6) (with 8.3 only) STC (8.2.1.4.3/ 8.3.8/8.4.8)	TDD

Table 1. Air interface nomenclature (From [10]).

b. Project Development Milestones

The first 802.16 standard, named IEEE Std 802.16-2001, was approved in December 2001 followed by two amendments which were the IEEE Std 802.16a and IEEE Std 802.16c. These were later superseded and made obsolete in 2004 by IEEE Std 802.16-2004. An amendment to the IEEE Std 802.16-2004 was concluded in 2005 with IEEE 802.16-2005 which addresses mobility. [13]

The works of this thesis is based on the active IEEE Std 802.16-2004 standard that supports fixed broadband wireless access.

3. IEEE 802.16 WirelessMAN-OFDM™ Physical Layer

The physical layer for the WirelessMAN-OFDM is defined in the IEEE 802.16-2004 standard, employs OFDM modulation and is designed for NLOS operation in the frequency bands below 11 GHz [10]. This section discusses the details of the IEEE 802.16 WirelessMAN-OFDM™ physical layer, as found in [10].

a. OFDM Symbol Description

The Inverse-Fast Fourier-Transform (IFFT) is used to create an OFDM waveform. A Cyclic Prefix (CP), which is a duplicate of the last section of the useful OFDM symbol, is required as shown in Figure 4 to protect against multipath interferences while maintaining the orthogonality of the tones. Defining G as the ratio of CP time to useful time (i.e. T_g/T_b), the standard specifies possible G values of 1/4, 1/8, 1/16 and 1/32. [10]

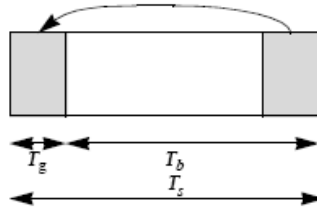


Figure 4. OFDM symbol time structure (From [10]).

An OFDM symbol (see Figure 5) is made up of two hundred and fifty-six subcarriers. This determines the FFT size to be used. Within an OFDM symbol, there are two hundred data subcarriers, eight pilot subcarriers, a DC null, twenty eight lower frequency guard subcarriers and twenty seven high frequency guard subcarriers. [10]

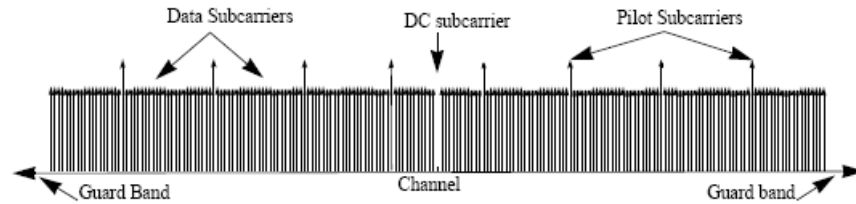


Figure 5. OFDM frequency description (From [10]).

The frequency offset indices of the subcarriers in an OFDM symbol are shown in Table 2.

Parameter	Value
Frequency offset indices of guard null subcarriers	-128,-127,...,-101 +101,+102,...,127
Frequency offset indices of pilot subcarriers	-88,-63,-38,-13,13,38,63,88
Frequency offset indices of data subcarrier	-100..-89,-87..-64,-62..-39,-37..-14,-12..-1 +1..12,14..37,39..62,64..87,89..100
Frequency offset indices of DC null subcarrier	0

Table 2. OFDM symbol parameters (From [10]).

b. Channel Coding

Channel coding specified in this standard comprises of, in the order for transmission, randomizing, forward error correction encoding and interleaving the data. For the receiver, the operations shall be applied in the reverse order. [10]

c. Randomization

Randomization shall be performed on each burst of data on the downlink and uplink except the preamble. Randomizing shall be reset with each OFDM symbol. If the amount of data to transmit does not fit exactly the amount of data allocated for an OFDM symbol, fixed with two hundred and fifty-six subcarriers, based on the selected

coding rate and modulation type, padding of logic ones shall be added to the end of the transmission block. For example, if the overall coding rate of 1/2 and a QPSK modulation are selected, the data allocated will be one hundred and eighty-four information bits. This is derived from one hundred and ninety-two data subcarriers allocated for an OFDM symbol. The QPSK modulation will allow two encoded bits per subcarrier. The encoding will restrict the randomized data to half the size of the encoder output. Of these randomized data, eight tail bits are to be reserved for the encoder to pad the data stream with logical zeroes. [10]

The pseudo random binary sequence (PRBS) generator as shown in Figure 6, shall be designed based on the polynomial $1 + X^{14} + X^{15}$. The shift-register of the randomizer shall be initialized with 100101010000000 for each OFDM symbol. Each data block to be transmitted shall enter sequentially into the randomizer, MSB first. [10]

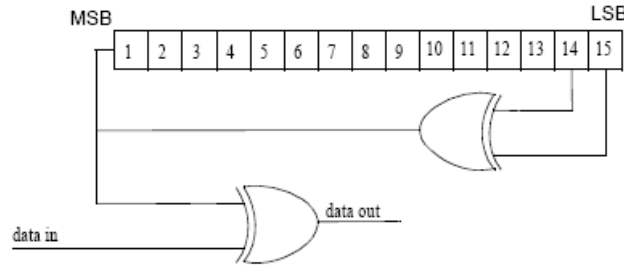


Figure 6. PRBS generator for data randomization (From [10]).

d. Forward Error Correction

The FEC specified in this standard consists of a concatenation of a Reed–Solomon outer code and a rate-adjustable convolutional inner code. At the transmitter, data shall first be encoded with the Reed–Solomon code before going through the convolutional encoder. [10]

The Reed–Solomon coding is not discussed here as it was not implemented in this thesis. For the convolutional encoder, it shall have a basic rate of 1/2 and a constraint length of seven. The generator is shown in Figure 7 where the generator polynomials for output X and Y are $1 + X + X^2 + X^3 + X^6$ and $1 + X^2 + X^3 + X^5 + X^6$, respectively. [10]

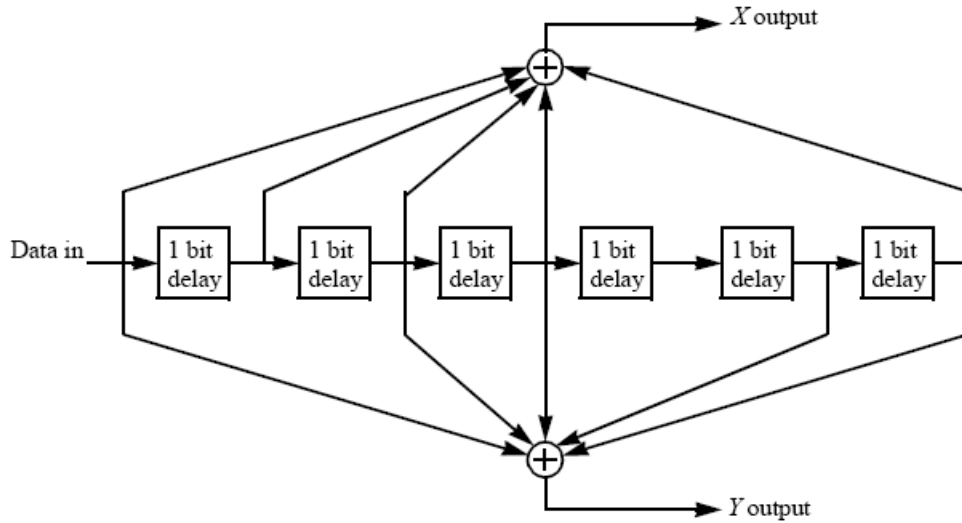


Figure 7. Convolutional encoder of rate 1/2 (From [10]).

Table 3 shows the puncturing patterns used to derive the different code rates. X precedes Y in the order of output. In the table, a “1” means a transmitted bit and “0” denotes a removed bit. For example, to achieve a code rate of 2/3, every third bit of the serial output stream is omitted for transmission. This equates to omitting the alternate bit of output X. [10]

	Code rates			
Rate	1/2	2/3	3/4	5/6
d_{free}	10	6	5	4
X	1	10	101	10101
Y	1	11	110	11010
XY	X_1Y_1	$X_1Y_1Y_2$	$X_1Y_1Y_2X_3$	$X_1Y_1Y_2X_3Y_4X_5$

Table 3. The inner convolutional code with puncturing configuration (From [10]).

Table 4 shows the block sizes used for the different modulations and code rates.

Modulation	Uncoded block size (bytes)	Coded block size (bytes)	Overall coding rate	RS code	CC code rate
BPSK	12	24	1/2	(12,12,0)	1/2
QPSK	24	48	1/2	(32,24,4)	2/3
QPSK	36	48	3/4	(40,36,2)	5/6
16-QAM	48	96	1/2	(64,48,8)	2/3
16-QAM	72	96	3/4	(80,72,4)	5/6
64-QAM	96	144	2/3	(108,96,6)	3/4
64-QAM	108	144	3/4	(120,108,6)	5/6

Table 4. Mandatory channel coding per modulation (From [10]).

e. Interleaving

Interleaving at the transmitter is carried out on all encoded data bits to guard against burst errors which may be uncorrectable by the FEC decoder at the receiver. Interleaving is done via two permutations on the incoming coded bits per OFDM symbol. Parameters necessary for computing the permutations are N_{cbps} which denotes the number of incoming coded bits per OFDM symbol, N_{cpc} which denotes the number of coded bits per subcarrier, and s which is derived from the computation, $\text{ceil}(N_{cpc}/2)$. N_{cpc} is dependent on the type of modulation and equates to 1, 2, 4 or 6 for BPSK, QPSK, 16-QAM, or 64-QAM modulation respectively. For a coded bit within a block of N_{cbps} bits, k represents its index order prior to the first permutation. After the second permutation, the index order of that same coded bit is denoted as m_k while j_k is the index after the second permutation. The interleaved data is then forwarded for modulation. [10]

The two-step permutation for interleaving are defined as follows:

$$(1) \quad m_k = (N_{cbps} / 12) \cdot k_{\text{mod}12} + \text{floor}(k / 12) \quad k = 0, 1, \dots, N_{cbps} - 1$$

$$(2) \quad j_k = s \cdot \text{floor}(m_k / s) + (m_k + N_{cbps} - \text{floor}(12 \cdot m_k / N_{cbps}))_{\text{mod}(s)}$$

$$k = 0, 1, \dots, N_{cbps} - 1$$

At the receiver, the de-interleaver carries out the reverse operations. A two-step permutation is used to re-order the coded bits after demodulation and before decoding. For a coded bit within a block of N_{cbps} bits, j represents its index order prior to the first permutation. After the second permutation, the index order of that same coded bit

is denoted as m_j while k_j is the index of that bit after the second permutation. The de-interleaved data is then forwarded for decoding. [10]

The two-step permutation for de-interleaving are defined as follows:

$$(1) \quad m_j = s \cdot \text{floor}(j / s) + (j + \text{floor}(12 \cdot j / N_{cbps}))_{\text{mod}(s)}$$

$$j = 0, 1, \dots, N_{cbps} - 1$$

$$(2) \quad k_j = 12 \cdot m_j - (N_{cbps} - 1) \cdot \text{floor}(12 \cdot m_j / N_{cbps})$$

$$j = 0, 1, \dots, N_{cbps} - 1$$

The first bit out of the interleaver shall map to the MSB in the constellation.

f. Modulation

After interleaving, the data bits are to the symbol mapper for modulation. The constellations for BPSK, Gray-mapped QPSK, 16-QAM, and 64-QAM are shown in Figure 8. The constellations shall be normalized with the indicated factor c to equalize average power. b_0 is the LSB. [10]

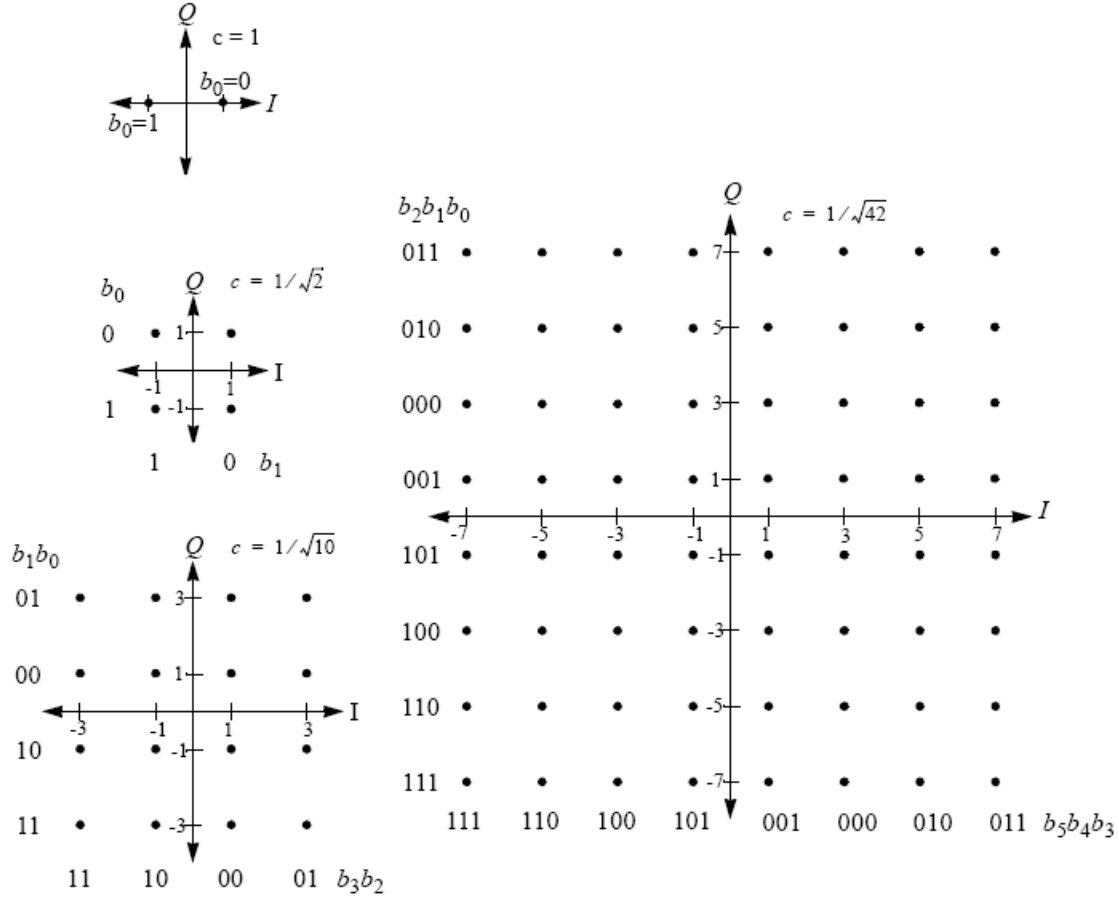


Figure 8. BPSK, QPSK, 16-QAM and 64-QAM constellations (From [10]).

The constellation-mapped data shall then be modulated onto the data subcarriers of an OFDM symbol in order of increasing frequency offset index via the inverse fast Fourier transform (IFFT). [10]

g. Pilot Modulation

Pilot subcarriers shall be inserted into the OFDM symbol for channel estimation. The PRBS generator, of the polynomial $X^{11} + X^9 + 1$, is shown in Figure 9 and is used to produce a sequence, w_k . The initialization sequences that shall be used on the downlink and uplink are shown in Figure 9 below. BPSK modulation is used for the pilot subcarrier. The value of each of the eight modulated pilot subcarrier within the

OFDM symbol of index k is derived from w_k as shown below. The first OFDM symbol is indexed as $k = 0$. [10]

(1) Downlink

$$c_{-88} = c_{-38} = c_{63} = c_{88} = 1 - 2w_k, \quad c_{-63} = c_{-13} = c_{13} = c_{38} = 1 - 2\overline{w_k}$$

(2) Uplink

$$c_{-88} = c_{-38} = c_{13} = c_{38} = c_{63} = c_{88} = 1 - 2w_k, \quad c_{-63} = c_{-13} = 1 - 2\overline{w_k}$$

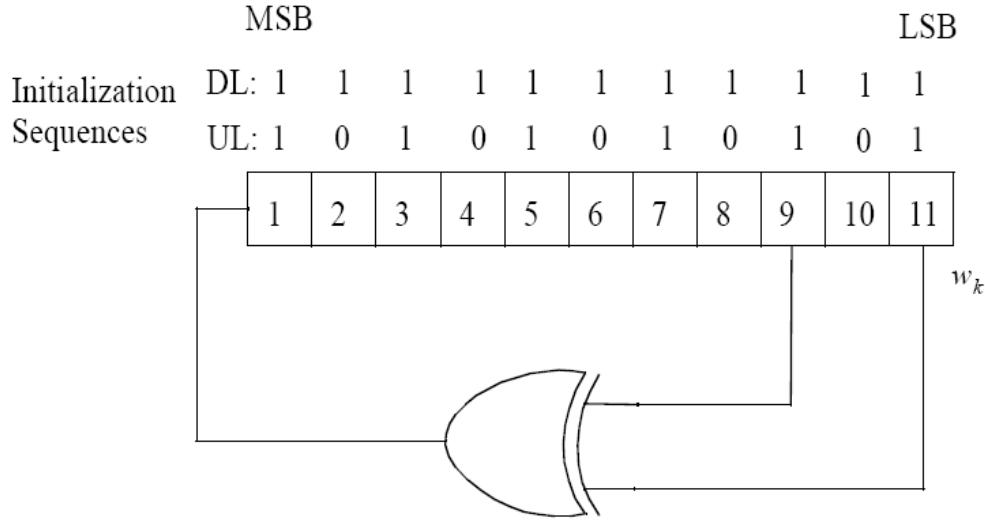


Figure 9. PRBS for pilot modulation (From [10]).

C. SUMMARY

This chapter presented the relevant background information that is useful for the understanding of the research. Key concepts presented were of the Software Defined Radio and the IEEE 802.16 standard. For the latter, the physical layer of the IEEE 802.16 WirelessMAN-OFDMTM was described in detail. The next chapter continues to present useful background information on the software design environment which includes the Software Communications Architecture and OSSIE.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SOFTWARE DESIGN ENVIRONMENT

A. SOFTWARE ARCHITECTURE

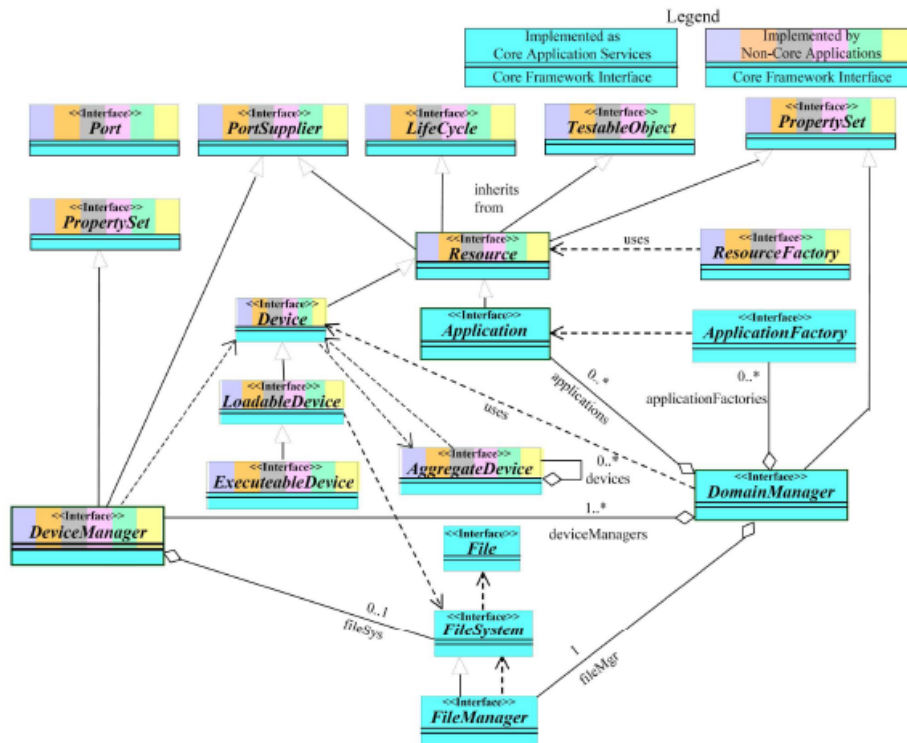
At the center of the SDR technology is the software architecture that governs the structure and operations within the SDR radio. The software architecture of a system details a collection of components and their interactions among each other [14].

1. Software Communications Architecture (SCA)

Many proprietary architectures exist, but to ensure portability and interoperability of the protocols on the different radios, an open architecture had to be developed. The Software Communications Architecture (SCA), developed by the US Department of Defense JTRS project, is such an architecture. While the SCA was originally intended solely for military use, it has gained commercial viability due to the efforts of groups like the Object Management Group (OMG) and the SDR Forum.

The Software Communications Architecture (SCA) is an open architecture framework that specifies the structure and operations within a SDR. It is a requirement specifications for the design of the SDR. The interfaces are defined by using the CORBA Interface Definition Language (IDL), and graphical representations are made by using Unified Modeling Language (UML) [16]. The operating environment consists of a Core Framework (CF), a CORBA middleware and an operating system.

The SCA Core Framework is illustrated in Figure 10. The figure shows the primary SCA interfaces. A software component communicates with other components via its interfaces. Definition of these interfaces in the SCA is based on the CORBA middleware which ensures compatibility of software components in terms of being able to communicate with each other. The benefit of using a middleware is explained in the next section.



2. Common Object Request Broker Architecture (CORBA)

Middleware is software that connects two or more software applications with non-compatible interfaces. With the middleware, different software applications written in different languages or running on different platforms can interoperate and communicate transparently. [18]

An example of a middleware is the Common Object Request Broker Architecture (CORBA) which is OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks.

In a general sense CORBA “wraps” around code written in some language, with a standard interface definition [19]. This greatly facilitates interoperability since the functional code is hidden and only takes care of computation. Two pieces of functional code written in different programming languages but “wrapped” with CORBA can now communicate with each other. Therefore CORBA provides the mechanism through which different software defined radio vendors can develop compatible software and hardware

interfaces. The users do not have to worry about downloading the correct software for the SDR since they should all be compatible.

B. OPEN SOURCE SCA IMPLEMENTATION::EMBEDDED (OSSIE)

OSSIE, developed by the Mobile and Portable Radio Research Group (MPRG), is an open source implementation of the SCA. OSSIE is a C++-based open source implementation of the SCA. Still a beta version release, the software also comes with a tool called the OSSIE Waveform Developer (OWD) for the rapid development of the SDR components and application waveforms. An evolving library of SDR components is also available on the MPRG's website. [7]

In this thesis, the SDR waveforms and components were built using OSSIE version 0.5.0.

C. WAVEFORM DEVELOPMENT

A development environment that automatically prototypes the code structure of the waveforms and components allows radio designers to better concentrate on the functional design of the component and waveform. A development environment also standardizes the code structure which makes it easier for subsequent modifications and improvement. Some Waveform Development Environments (WDE) feature automatic code generation where a skeleton code is generated, with the functional code to be added on. This makes developing a SDR easier since a developer needs only to concentrate on the functional code design. [20]

There are several WDEs commercially available on the market today that deals specifically with the SCA. However, unlike the OSSIE Waveform Developer (OWD), they are all proprietary tools. Table 5 summarizes the major features supported by some of the commercially off-the-shelf (COTS) available SCA WDEs and compares them to OWD. [20]

Software Package	XML Generation	Code Generation	Domain Management	Free
Harris dmTK	Yes	No	Yes	No
Zeligsoft Component Enabler	Yes	Yes	No	No
CRC Development Toolset	Yes	Yes	No	No
PrismTech Spectra	Yes	Yes	No	No
OSSIE Waveform Developer	Yes	Yes	No	Yes

Table 5. SCA development environment comparison (From [20]).

D. OSSIE WAVEFORM DEVELOPER (OWD)

The OSSIE Waveform Developer is a form of Graphical User Interface (GUI) that facilitates the designing of SCA waveforms and components. In addition, the OWD also generates skeleton C++ code and the utility files necessary to install the components and waveforms as well as run the application waveform. [20]

The OWD facilitates creation of new SDR components as well as building of a waveform application using components available in the stored library. In using OWD to create new SDR components, the basic structure of the component can be designed within the OWD which will then generate the skeleton C++ code for the component. The developer can then proceed to fill in the skeleton code with the desired functionality of that component without having to worry about the SCA-compliant interfaces. [20]

1. File and Directory Structure

Figure 11 shows a typical directory and file layout resulting from waveform generation using OWD [20].

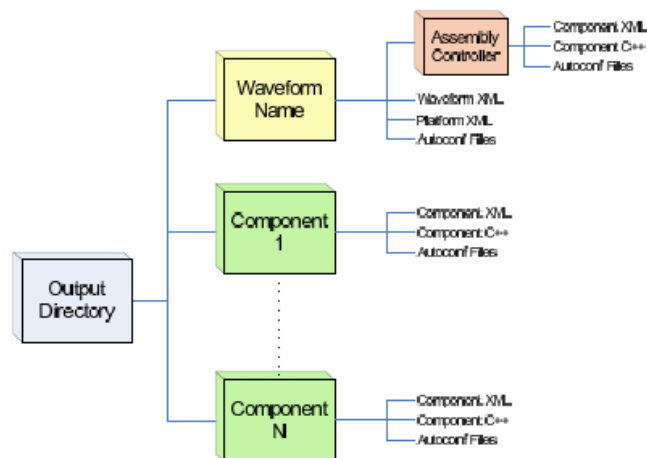


Figure 11. OWD generated directory layout (From [20]).

Tables 6 and 7 list the files that are generated for typical waveforms and components respectively [20].

File	Type
DomainManager.dmd.xml	SCA Waveform XML
DomainManager.spd.xml	SCA Waveform XML
DomainManager.scd.xml	SCA Waveform XML
DomainManager.prf.xml	SCA Waveform XML
DeviceManager.dcd.xml	SCA Waveform XML
DeviceManager.spd.xml	SCA Waveform XML
DeviceManager.scd.xml	SCA Waveform XML
DeviceManager.prf.xml	SCA Waveform XML
<Waveform Name>.sad.xml	SCA Waveform XML
<Waveform Name>_DAS.xml	OSSIE Waveform XML
configure.ac	Autoconf File
Makefile.am	Autoconf File
reconf	Autoconf File
aclocal.d	Autoconf Directory

Table 6. OWD generated waveform files (From [20]).

File	Type
<Component Name>.spd.xml	SCA Component XML
<Component Name>.scd.xml	SCA Component XML
<Component Name>.prf.xml	SCA Component XML
<Component Name>.h	OSSIE Component C++
<Component Name>.cpp	OSSIE Component C++
main.cpp	OSSIE Component C++
port_impl.h	OSSIE Port Implementation C++
port_impl.cpp	OSSIE Port Implementation C++
configure.ac	Autoconf File
Makefile.am	Autoconf File
reconf	Autoconf File
aclocal.d	Autoconf Directory

Table 7. OWD generated component files (From [20]).

2. XML Domain Profile

The XML files that are generated with each waveform and component are integral to the operation of the radio. As described by the JTRS JPEO, these files “describe the identity, capabilities, properties, and interdependencies of the hardware devices and software components that make up the system” [17].

3. C++ Code

As shown in Table 7, there are five C++ files that are auto-generated for each new component. These files make up the functionality of a particular component. The developer needs only to modify these files so as to add the necessary function intended of the component. The following describes in further detail these files except the `main.cpp` file which contains default utility code transparent to the developer and not critical to the component design in terms of code modifications to add component functionality.

a. Component Name.h

This is a C++ header file with the same name as the component and contains all of the C++ class definitions for the particular component. Member functions and port declarations compliant to the SCA are included by default. Necessary constant declarations representing parameters of the component function can also be included in this file.

b. Component Name.cpp

This is a C++ implementation file. By default, this file includes the basic SCA functionality such as *getPort*, *start*, *stop*, *releaseObject*, the constructor and destructor for the component. The port interfaces of the component are also instantiated. Very importantly, this is also the file for the developer to add functions that defines the SDR component. Member functions can be added to process data according to that component functionality of the SDR.

c. port_impl.h

This file contains C++ class definitions for each type of port interface used in the component. Two classes are automatically generated for each port interface on a component. They are *dataIn_<Interface Name>_i* and *dataOut_<Interface Name>_i*. These are classes that define operations for port communications between components.

d. port_impl.cpp

This file contains the C++ implementation of the member functions defined in the `port_impl.h` file.

E. SUMMARY

Key concepts presented in this chapter were of the Software Communications Architecture and OSSIE. For the latter, the OSSIE Waveform Developer was described in detail. The next chapter will present the process of software development. This includes the approach and considerations taken before elaborating on the design of the waveform and components.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. SOFTWARE DEVELOPMENT

A. APPROACH

As the works of this thesis were also the first attempts to use OSSIE to implement SDR of the IEEE 802.16 standard, there were little or no specific resources to use as references. The research developed the software radio components from scratch. Thus, a software process model was adopted to structure the development of the software components.

1. Incremental Development Model

The intent is to develop the application waveform, which comprises of SDR components, incrementally and systematically. The process starts with a simple implementation of a subset of the software requirements, in this case the SDR components, and iteratively enhances the evolving versions until the full system, i.e. the SDR application waveform, is implemented. [21]

The incremental development model comprises of three stages: Design, Develop and Verify. Figure 12 describes the interrelationship between these three stages as a model and how it corresponds to processes in the software waveform development of the IEEE 802.16-2004 standard. [22]

a. Design

This stage starts with defining the outline software requirements and assigning these requirements to the specific increment. Specifically, the overall waveform design shall be addressed conceptually with respect to the IEEE 802.16-2004 standard requirement. The conceptual waveform design will then be decomposed into smaller and fundamental components. These components will be the increments of the model. Component design will then be addressed from the simplest to the complicated. In this way, the system software architecture is designed and shall serve as a framework for actual software development at the next stage.

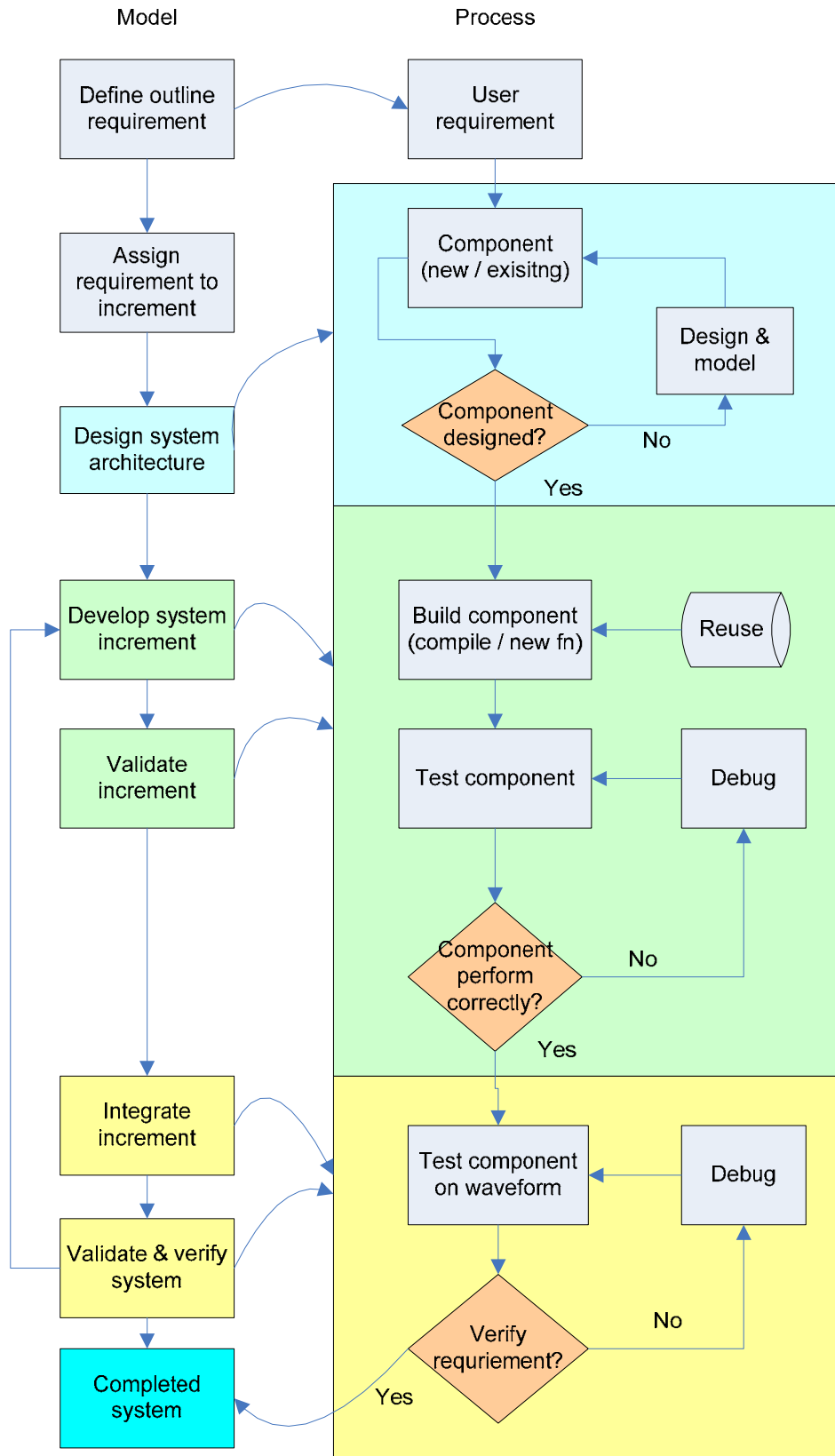


Figure 12. Incremental Development Model (After [22]).

b. Develop

This is the actual work of software development and programming, whereby the system requirements and pseudo-codes are converted to actual software languages. The coded algorithms are validated incrementally to ensure they meet the functionality expectations. Successful increments are stored for future use and new functionalities through design modifications will be introduced for the next increment.

c. Verify

With the incremental development model, the software system design gets larger and more complex with each iteration. Increments will be integrated in this stage and the system as a whole will be verified to meet the holistic software requirements. For this research, the eventual completed system must be able to emulate the IEEE 802.16 WirelessMAN-OFDMTM physical layer.

B. CONSIDERATIONS

Several considerations drive the design of the SDR components. These considerations are translated from the objectives of this thesis. The following sections describe these considerations and the decisions that drive the subsequent design of the SDR components.

1. Assumptions

The SDR, though largely implemented in software still require a hardware platform to run the software as well as provide for the RF front-end of the system. As the main focus of the thesis objectives is to create SDR components that will contribute to the library of components for further development work, speed was not important. Therefore, the SDR components designed using OSSIE needed only to be run on any general purpose processor.

The interface to the hardware RF front-end for actual air transmission and reception is set aside as future work. Nevertheless, an assumption has to be made on the RF front-end architecture to facilitate the design of the SDR waveform. Figure 13

illustrate the assumed SDR architecture design consisting of both the hardware-implemented RF front-end and the software-implemented SDR waveform.

2. Functionality

The IEEE 802.16 standard encompasses many features over a wide range of frequencies to ensure a high quality of service. This thesis concentrates on the IEEE 802.16 WirelessMAN-OFDMTM physical layer standard which uses licensed bands from 2 to 11GHz. The goal of the thesis is to implement SDR components for a basic single-mode configuration of the physical layer standard. These were explained in Chapter II of this thesis. In addition, no subchannelizations were considered. In other words, the full bandwidth would be used. The transceiver design considered was to be for the Subscriber Station (SS).

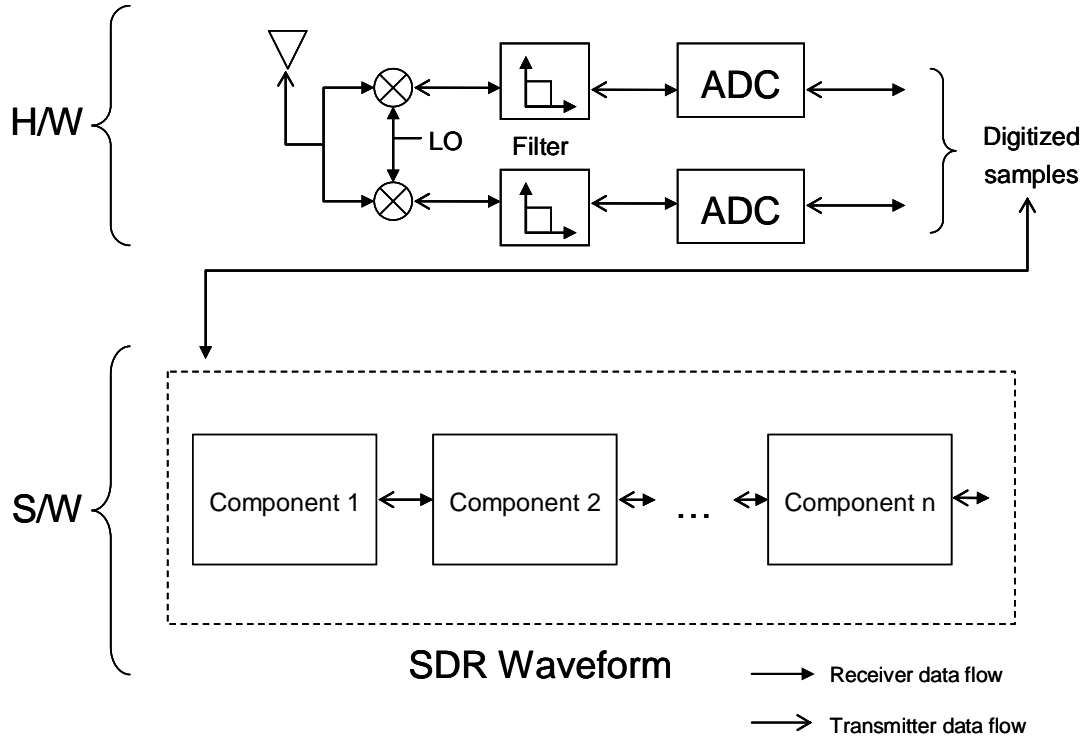


Figure 13. Assumed SDR architecture design.

3. Reusability and Reconfigurability

A very important goal of the thesis is that the implemented SDR components be reusable so that the components can be contributed to a growing active library. The components should be able to be reused, with little or no amendments, for building other waveforms. This requires the waveform to be broken down to as many simple and elementary components as possible. This way, the components can be designed in a way as generic as possible such that it is single-functioned and not customized to any waveform standard or profile. The components should also be designed in a way where their numbers of interfaces are minimized and operations kept simple and rudimentary. In the case where amendments to the component design are required, the code should be well-documented so that effort to amend the code can be made easy.

Achieving the above will also enable easy reconfiguration of the waveform. The waveform can be reconfigured easily without having drastic reconstruction by simply replacing the generic components with others from the library. In other words, there should not be any need to build another component with functions customized to the new configuration when a component with the same fundamental function already exists..

4. Constraints

Firstly, the SDR waveforms and components were built in this thesis using OSSIE version 0.5.0 which was still a beta version under development. Thus it was important that there be proper documentation to track the OSSIE version upon which the code for the SDR components were based. There was a given possibility that code developed based on previous beta versions of OSSIE may not run on newer beta versions since, at the developmental stage, the push for an effective final product outweighs the need to maintain backward compatibility.

Secondly, OSSIE 0.5.0 does not allow a single port interface of a component to have multiple connections to other components. It's strictly a one-to-one connection. For example, Component A cannot be connected via a single output port to inputs of Component B and C concurrently. Component A needs to have two output ports to be able to connect to Component B and C.

Thirdly, although OSSIE 0.5.0 allows multiple input port interfaces of the same type, it should be noted that a common buffer is used to store data transacted through these ports. For example, Component A may have two input ports of type *realShort*. However, data received through these two ports are stored in a common buffer and runs the risk of overwriting each other. It is thus important to make use of mutexes, which is a programming variable used to lock resources to prevent sharing, or ensure proper usage synchronization to avoid unintended data erasure [23].

Fourthly, as the interface to the hardware RF front-end will not be implemented in this thesis, there is a need to circumvent such that testing of the developed SDR components will still be effective.

C. WAVEFORM DESIGN

To achieve reusability of the components and easy reconfigurability of the waveform, the waveform has to be composed of many simple and elementary components. As such, the transmitter and receiver waveform were designed where the working model to be implemented was translated from the conceptual model shown in Figure 14. The waveform design of the transmitter and receiver working models are shown respectively in Figure 15 and Figure 16.

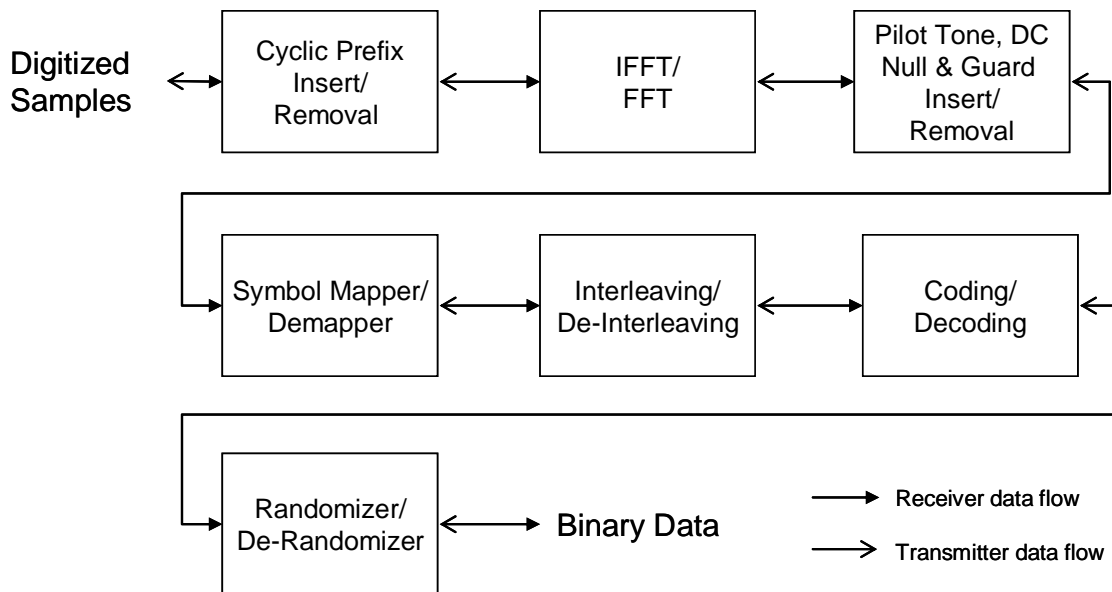


Figure 14. Conceptual waveform design.

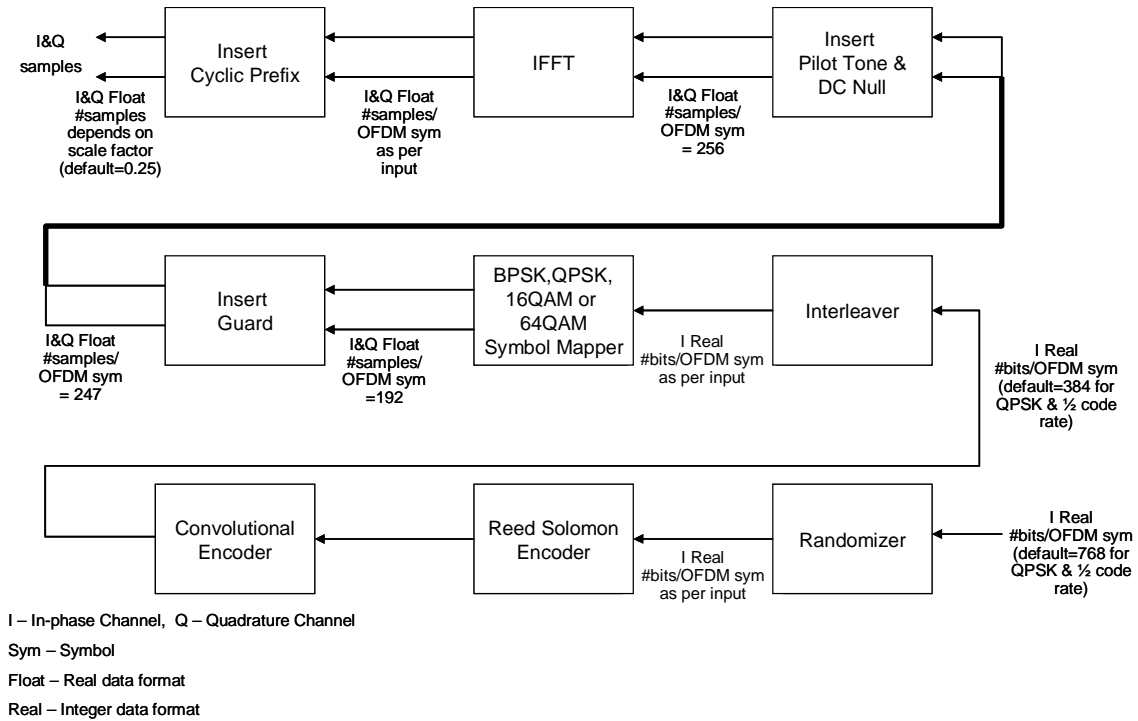


Figure 15. Working transmitter waveform design.

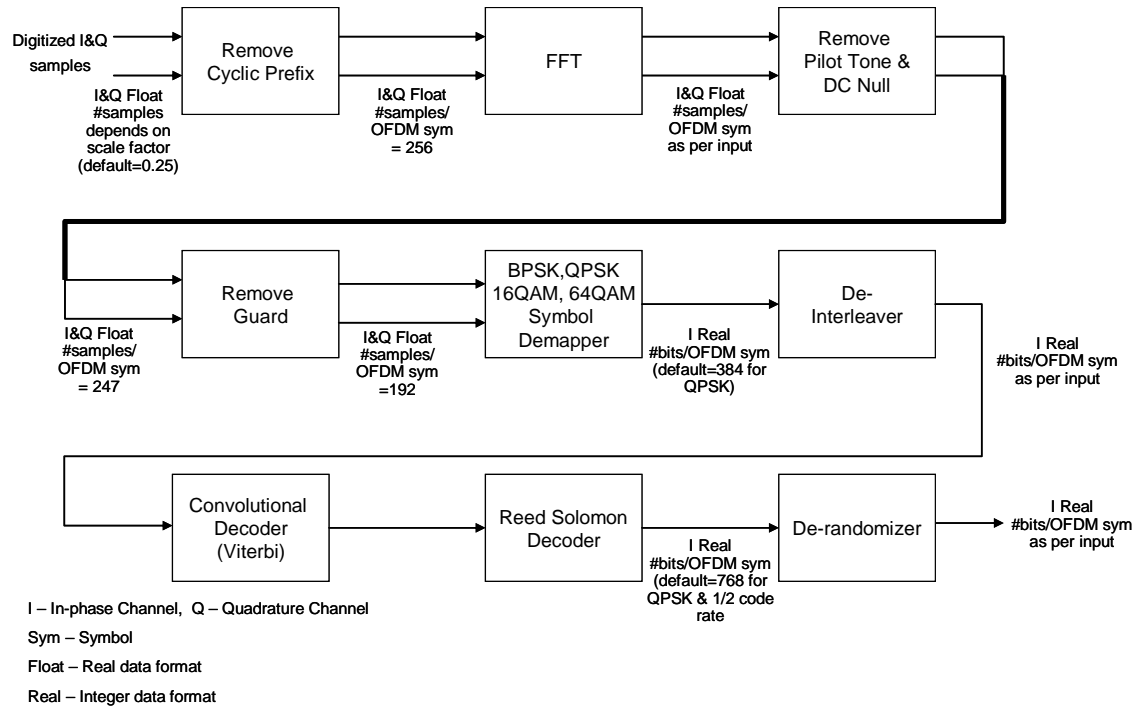


Figure 16. Working receiver waveform design.

D. COMPONENT DESIGN

Every component in the software architecture is structured somewhat similarly by the automatic code generation feature of the OWD. As described in Chapter III, subsequent modification to customize the auto-generated code to the intended component function involves only the four C++ files. Figure 17 illustrates the relationship with respect to the component functionality design between these four files.

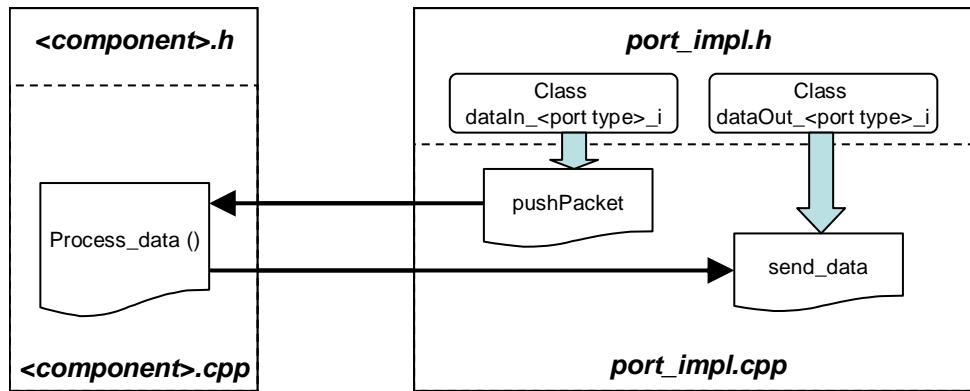


Figure 17. Functional architecture of the OWD auto-generated C++ code.

The function *process_data* carries out the main component function and is to be added into the auto-generated code. This is where incoming data from preceding component is processed before sending it off to the succeeding component. The functions *pushpacket* and *send_data* in the *port_impl.cpp* file provide the input and output services for process data.

Several port types are available for use of which four were applicable in the component design in this thesis. Components with a single channel port interface transferring data of the type integer i.e., whole numbers only, will use the *realShort* port type. If the data is of the type real, the component will use the *realFloat* port type. Components with a dual channel port interface transferring data of the type integer i.e., whole numbers only, will use the *complexShort* port type. If the data on each of the two channel interfaces is of the type real, the component will use the *complexFloat* port type.

In SDR design, the dual channel port interfaces facilitate data flow in the In-phase and Quadrature channels.

1. Transmitter

The fundamental components required in the transmitter waveform are illustrated in Figure 15. As with the approach of the software development using the Incremental Development model, the simplest components were developed first and added as increments to the waveform before moving on to develop more complicated components.

The following sections describe the design of the components within the transmitter waveform with the exception of the Reed-Solomon Encoder which was not developed within this thesis work. The component to implement the IFFT was reused and modified slightly from the one developed by Leong Wai Kiat, who was working on the IEEE 802.11a implementation using OSSIE for his thesis [22].

a. Randomizer

Component name: *randomizer_802_16*

Port design: *randomizer_802_16* has one input and one output port. Both ports are of type *realShort*.

Functional design: Firstly, incoming data to the component shall be padded up with logic ones at the end of the data block, if it does not fully fit the amount of data allocated to form an OFDM symbol subsequently which is fixed with two hundred and fifty-six subcarriers. The amount of data allocated depends on the code rate and modulation scheme selected for the waveform. This function is embedded in the *pushpacket* function in `port_impl.cpp`. The data shall then be randomized accordingly as described in Chapter II. The described functions are illustrated in Figure 18.

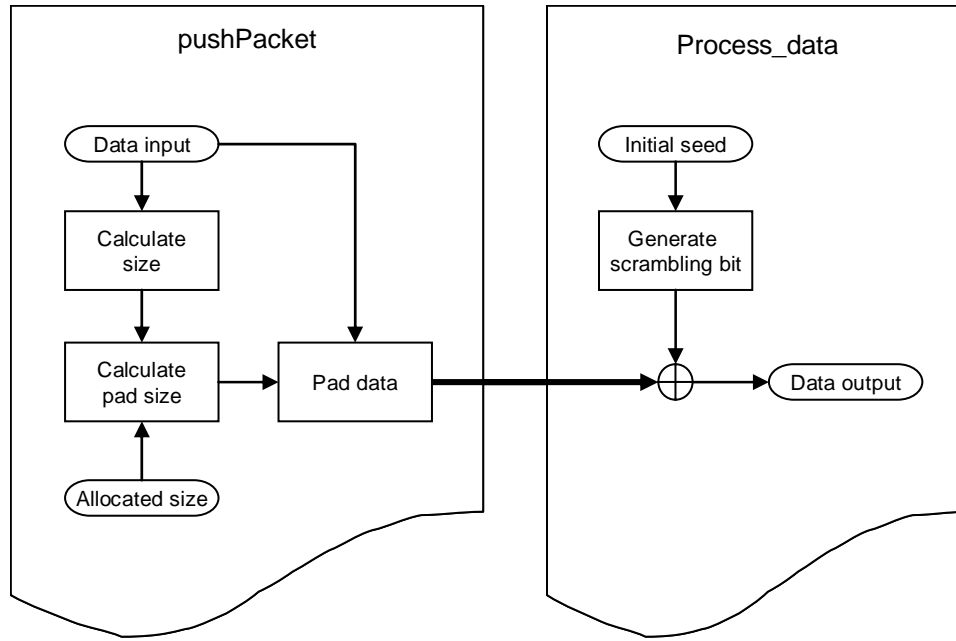


Figure 18. Functional description of component *randomizer_802_16*.

b. Convolutional Encoder

Component name: *CC_encode_802_16*

Port design: *CC_encode_802_16* has one input and one output port. Both ports are of type *realShort*.

Functional design: The data block shall be convolutionally coded as described in Chapter II, with a rate of 1/2, 2/3, 3/4 or 5/6 corresponding to the selected waveform profile. The encoder is designed based on a fundamental 1/2 code rate. Higher rates are derived from it by puncturing the encoded data. The described function is illustrated in Figure 19.

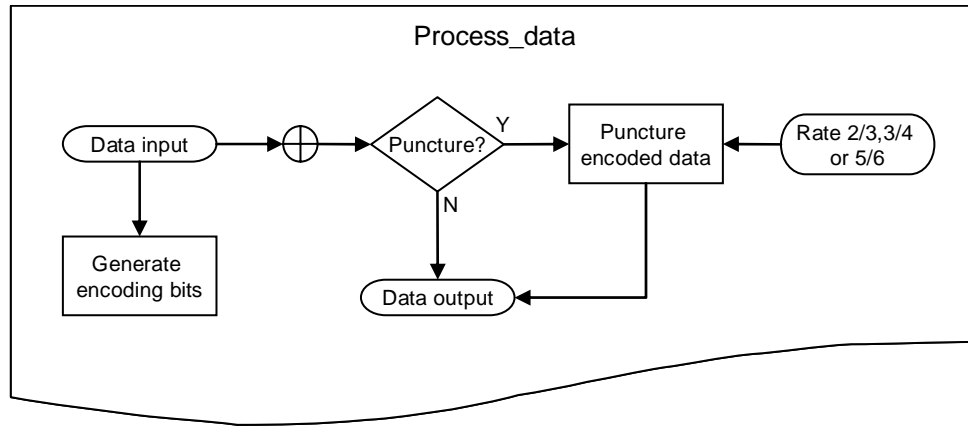


Figure 19. Functional description of component *CC_encoder_802_16*.

c. Interleaver

Component name: *interleaver_802_16*

Port design: *interleaver_802_16* has one input and one output port. Both ports are of type *realShort*.

Functional design: The data block shall be interleaved as described in Chapter II. The interleaver re-arranges the order of the incoming data using two permutations. The described function is illustrated in Figure 20.

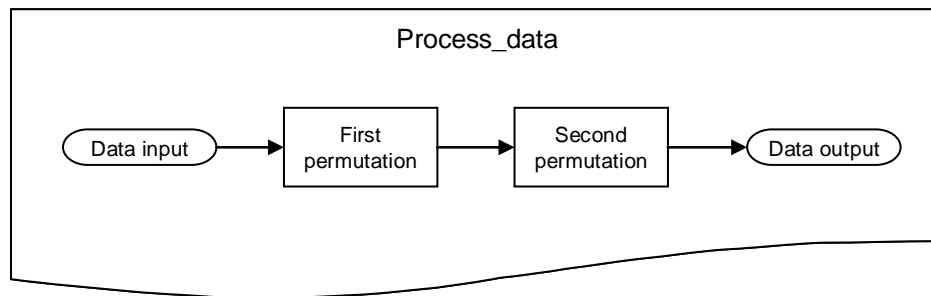


Figure 20. Functional description of component *interleaver_802_16*.

d. BPSK Symbol Mapper

Component name: *bpsk_mod_802_16*

Port design: *bpsk_mod_802_16* has one input port of type *realShort* and one output port of type *ComplexFloat*.

Functional design: The data block shall be BPSK modulated based on the constellation shown in Chapter II. To be consistent with the format of the other mapping schemes, two output ports for the In-phase and Quadrature channels are provided for in the structure although only the In-phase channel is necessary. For this component, the incoming data will always map to zero on the Quadrature channel. The described function is illustrated in Figure 21.

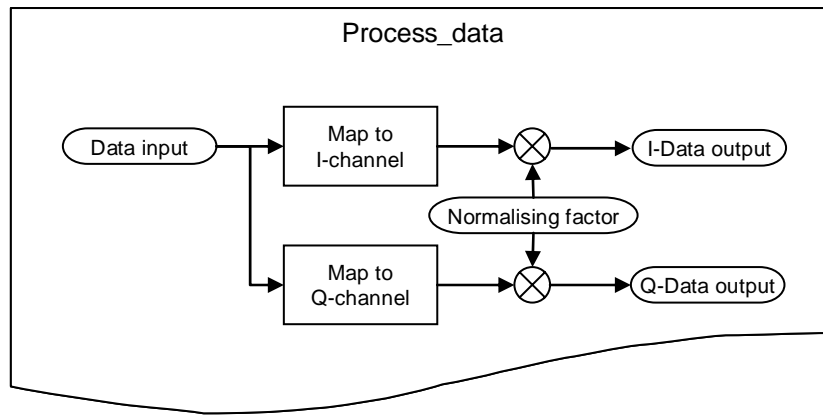


Figure 21. Functional description of component *bpsk_mod_802_16*.

e. QPSK Symbol Mapper

Component name: *qpsk_mod_802_16*

Port design: *qpsk_mod_802_16* has one input port of type *realShort* and one output port of type *ComplexFloat*.

Functional design: The data block shall be QPSK modulated based on the constellation shown in Chapter II. The described function can be similarly illustrated as in Figure 21.

f. 16-QAM Symbol Mapper

Component name: *QAM16_mod_802_16*

Port design: *QAM16_mod_802_16* has one input port of type *realShort* and one output port of type *ComplexFloat*.

Functional design: The data block shall be 16-QAM modulated based on the constellation shown in Chapter II. The described function can be similarly illustrated as in Figure 21.

g. 64-QAM Symbol Mapper

Component name: *QAM64_mod_802_16*

Port design: *QAM64_mod_802_16* has one input port of type *realShort* and one output port of type *ComplexFloat*.

Functional design: The data block shall be 64-QAM modulated based on the constellation shown in Chapter II. The described function can be similarly illustrated as in Figure 21.

h. Insert Guard Subcarriers

Component name: *guardIns_802_16*

Port design: *guardIns_802_16* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: The data block shall have null data, equivalently subcarriers, appended to its front and end as described in Chapter II. The described function is illustrated in Figure 22.

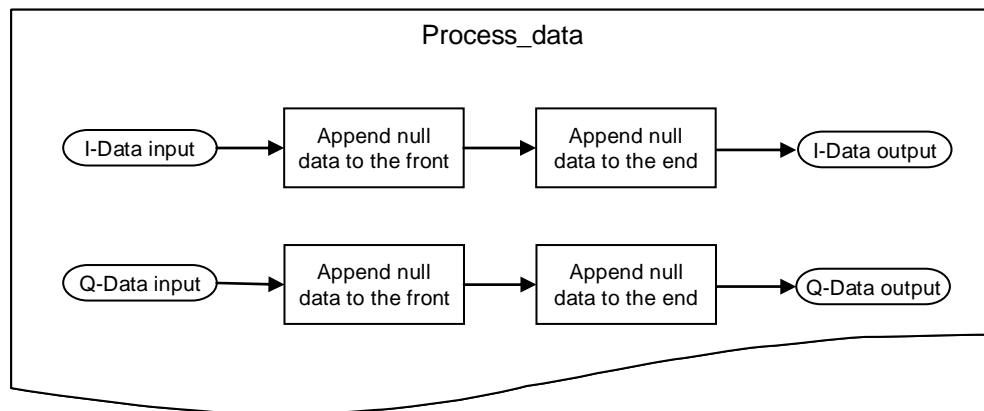


Figure 22. Functional description of component *guardIns_802_16*.

i. Insert Pilot Tone and DC Null Subcarriers

Component name: *ptIns_802_16*

Port design: *ptIns_802_16* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: The data block shall have pilot tone and DC null data, equivalently subcarriers, inserted into prescribed locations of the incoming data block as described in Chapter II. The values of the pilot tone data are to be determined through a randomizing function which defers between the uplink and downlink transmission and has to be pre-selected. The described function is illustrated in Figure 23.

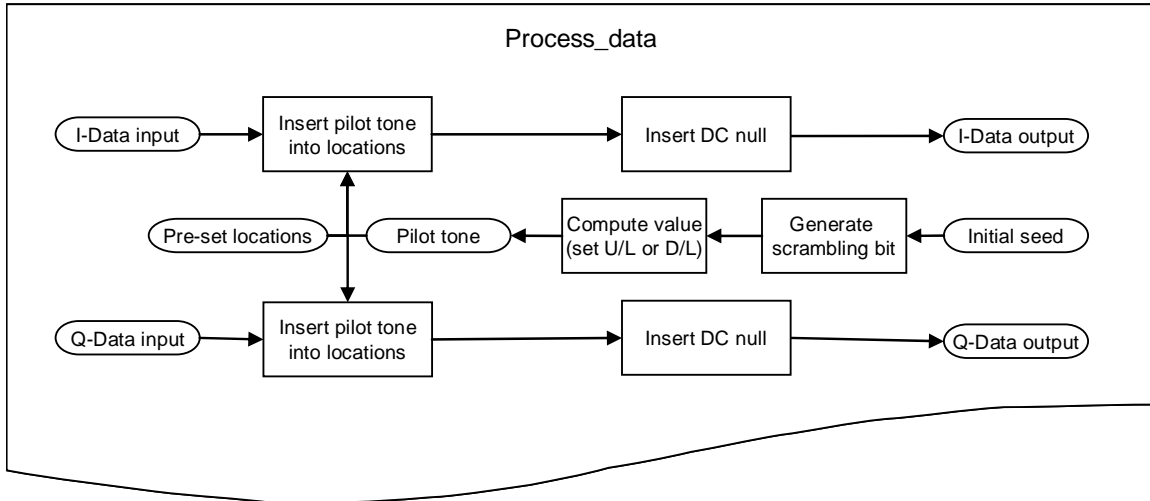


Figure 23. Functional description of component *ptIns_802_16*.

j. Inverse Fast Fourier Transform (IFFT)

Component name: *Data_IFFT*

Port design: *Data_IFFT* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: It implements the Inverse Fast Fourier Transform for OFDM modulation. The input data are taken as frequency samples and converted to time samples using the Decimation-In-Time (DIT) Permuted Input - Natural Output (PINO)

IFFT algorithm, which is described by Leong in [22]. The described function is illustrated in Figure 24 below.

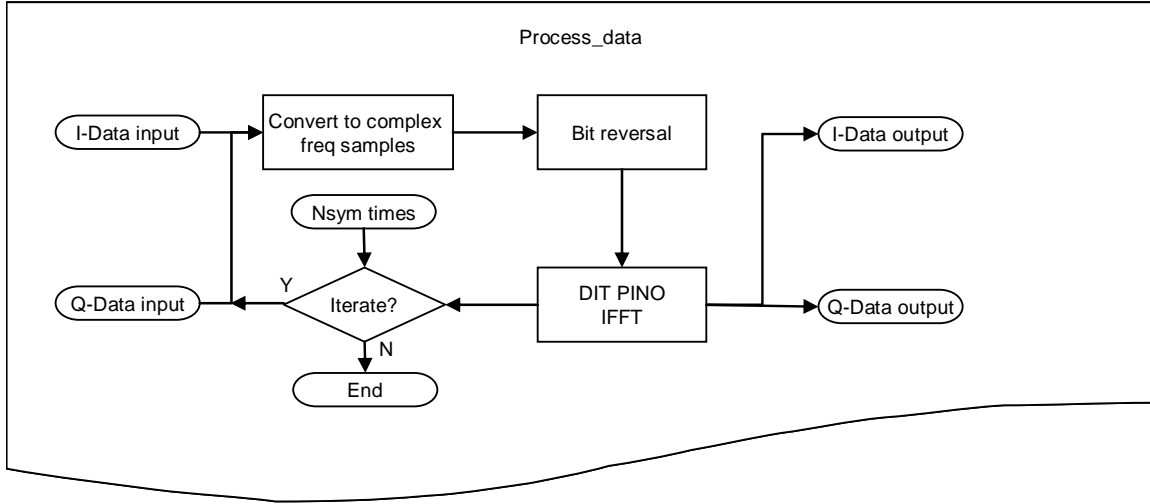


Figure 24. Functional description of component *Data_IFFT* (After [22]).

k. Insert Cyclic Prefix

Component name: *cpIns_802_16*

Port design: *cpIns_802_16* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: The data block shall have a cyclic prefix, equivalently a block of duplicated subcarriers, appended to its front as described in Chapter II. The proportion of the data to duplicate and append is to be pre-set. The described function is illustrated in Figure 25.

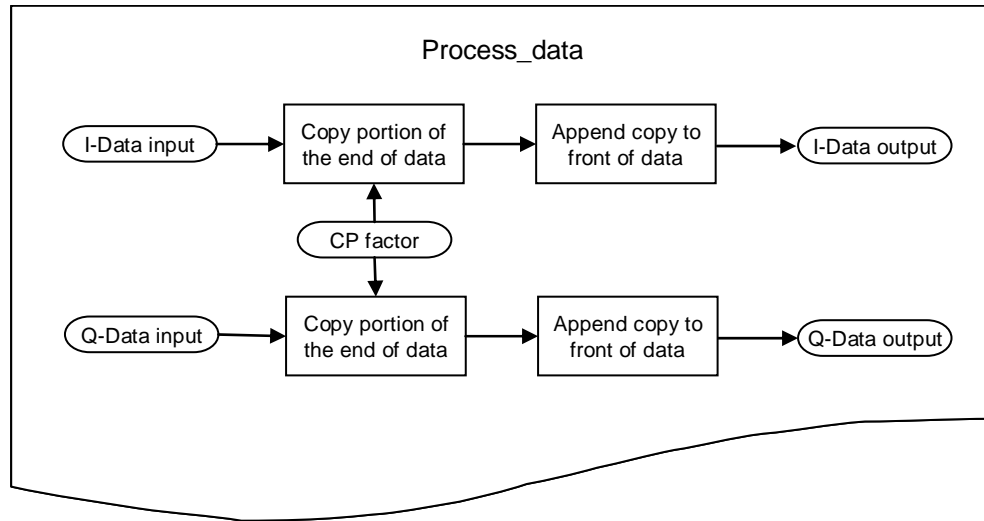


Figure 25. Functional description of component *cpIns_802_16*.

2. Receiver

The fundamental components required in the receiver waveform are illustrated in Figure 16. As with the approach of the software development using the Incremental Development model, the simplest components were developed first, in step with the associated transmitter component, and added as increments to the waveform before moving on to develop more complicated components.

The following sections describe the design of the components within the receiver waveform with the exception of the Reed-Solomon Encoder which was not implemented within this thesis work. The components to implement the Fast Fourier Transform and Convolutional decoding were reused and modified slightly from the ones developed by Leong Wai Kiat who was working on the IEEE 802.11a implementation using OSSIE for his thesis [22].

a. *De-randomizer*

Component name: *derandomizer_802_16*

Port design: *derandomizer_802_16* has one input and one output port.

Both ports are of type *realShort*.

Functional design: The incoming scrambled data shall be de-randomized by applying the same randomizing methodology as described in Chapter II. The described functions are illustrated in Figure 26.

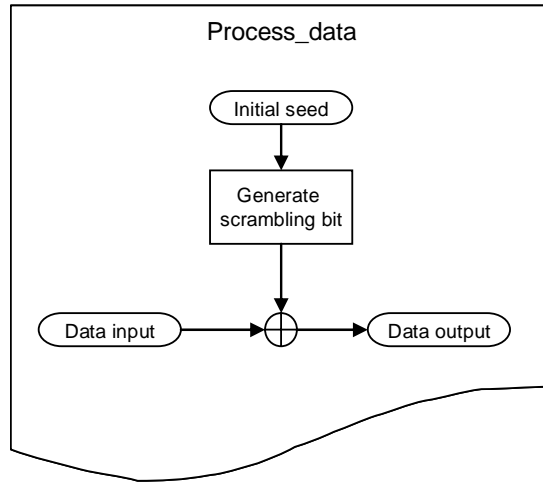


Figure 26. Functional description of component *derandomizer_802_16*.

b. Convolutional Decoder

Component name: *DATA_conv_dec*

Port design: *DATA_conv_dec* has one input and one output port. Both are of type *realShort*.

Functional design: Viterbi decoding is chosen to decode the deinterleaved bit streams of convolutional codes. The incoming data have been convolutionally encoded with a rate of 1/2, 2/3, 3/4 or 5/6. Except in the case of code rate 1/2, dummy bits are inserted prior to decoding since the higher code rates are derived from the basic 1/2 code rate by puncturing the encoded data at the transmitter. The functional flow is shown in Figure 27, 28 and 29. [22]

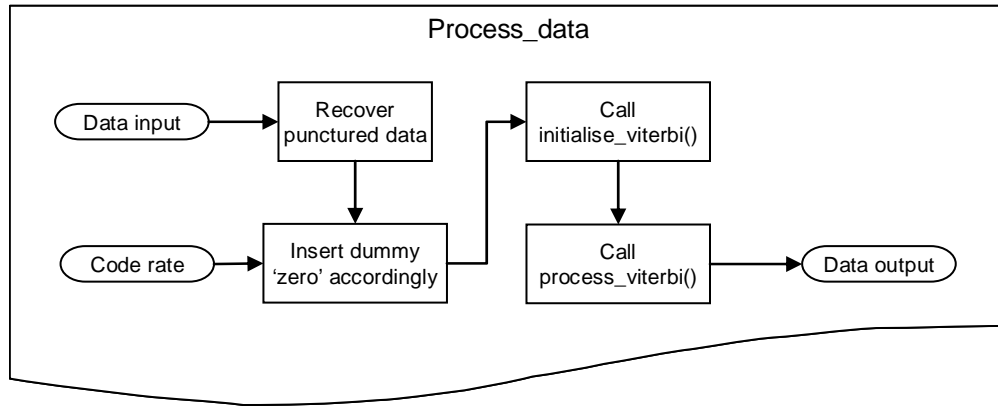


Figure 27. Functional description of component *Data_conv_dec* (After [22]).

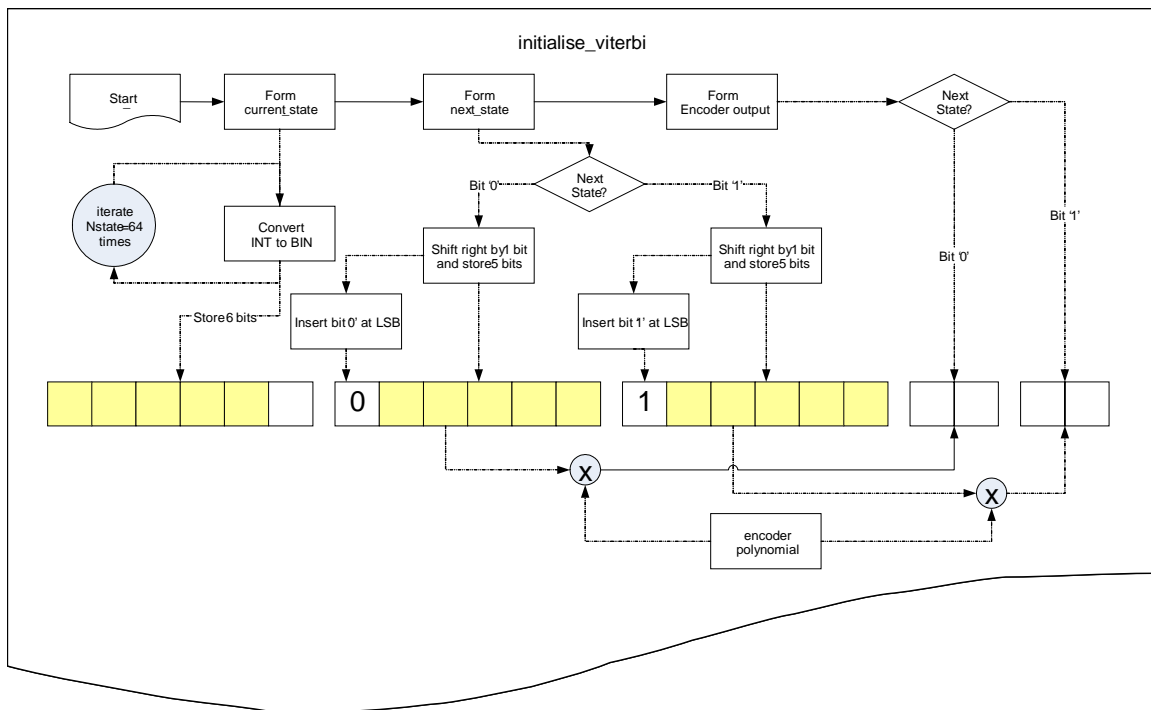


Figure 28. Functional description of function *initialize_viterbi* within Component *Data_conv_dec* (From [22]).

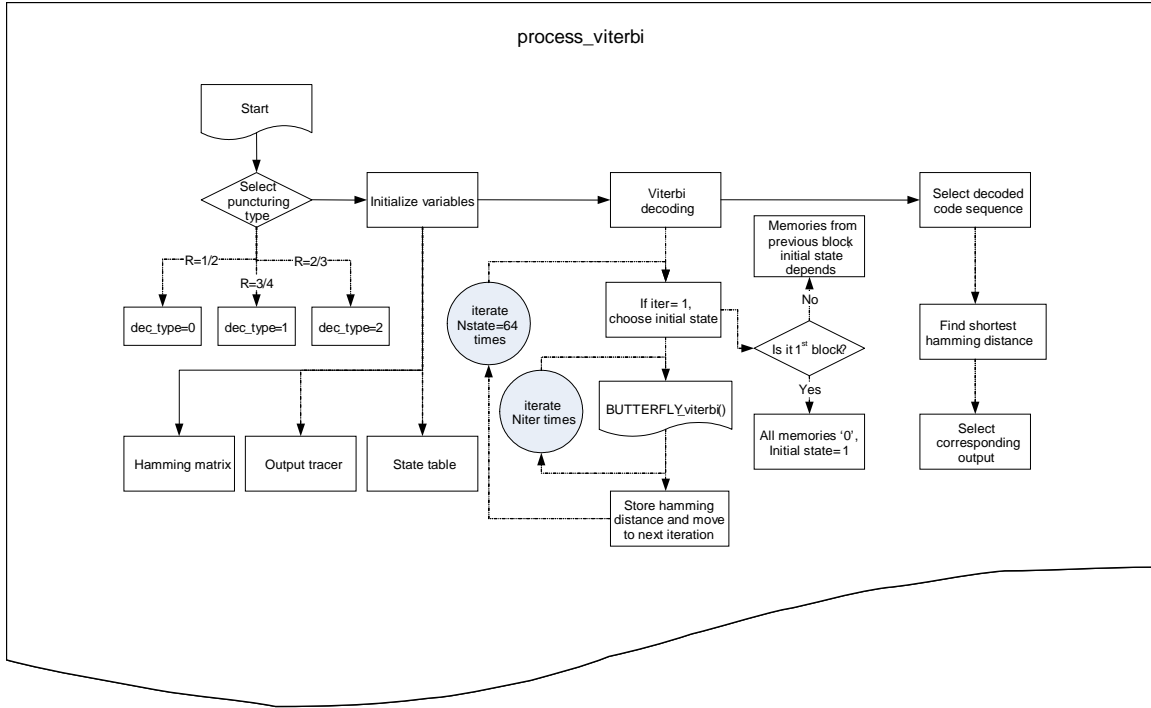


Figure 29. Functional description of function *process_viterbi* within component *Data_conv_dec* (From [22]).

c. De-interleaver

Component name: *deinterleaver_802_16*

Port design: *deinterleaver_802_16* has one input and one output port. Both ports are of type *realShort*.

Functional design: The data block shall be de-interleaved as described in Chapter II. The de-interleaver re-arranges the order of the incoming data using two permutations. The described function is illustrated in Figure 30.

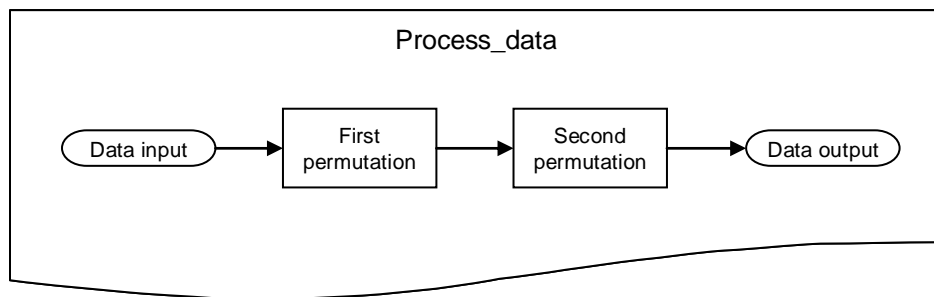


Figure 30. Functional description of component *deinterleaver_802_16*.

d. BPSK Symbol De-mapper

Component name: *bpsk_demod_802_16*

Port design: *bpsk_demod_802_16* has one input port of type *ComplexFloat* and one output port of type *realShort*.

Functional design: The data block shall be BPSK demodulated based on the constellation shown on Figure 8 in Chapter II. To be consistent with the format of the other mapping schemes, two inputs for the In-phase and Quadrature channels are catered for in the structure although only the In-phase channel is necessary. For this component, the algorithm will only process data from the In-phase channel. The described function is illustrated in Figure 31.

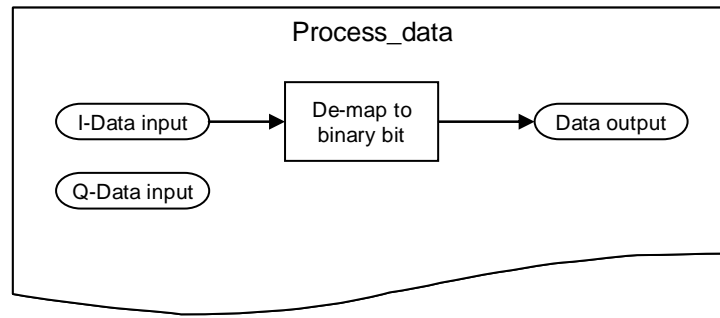


Figure 31. Functional description of component *bpsk_demod_802_16*.

e. QPSK Symbol De-mapper

Component name: *qpsk_demod_802_16*

Port design: *qpsk_demod_802_16* has one input port of type *ComplexFloat* and one output port of type *realShort*.

Functional design: The data block shall be QPSK demodulated based on the constellation shown on Figure 8 in Chapter II. The described function is illustrated as in Figure 32.

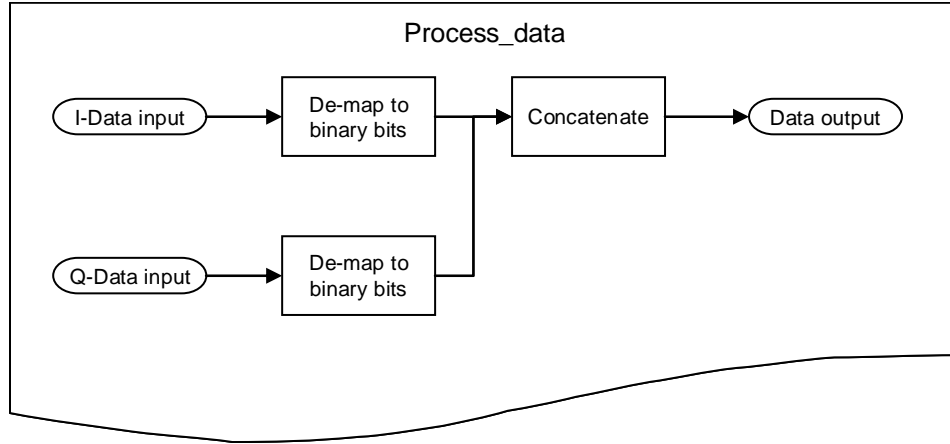


Figure 32. Functional description of component *qpsk_demod_802_16*.

f. 16-QAM Symbol De-mapper

Component name: *QAM16_demod_802_16*

Port design: *QAM16_demod_802_16* has one input port of type *ComplexFloat* and one output port of type *realShort*.

Functional design: The data block shall be 16-QAM demodulated based on the constellation shown on Figure 8 in Chapter II. The described function can be similarly illustrated as in Figure 32.

g. 64-QAM Symbol De-mapper

Component name: *QAM64_demod_802_16*

Port design: *QAM64_demod_802_16* has one input port of type *ComplexFloat* and one output port of type *realShort*.

Functional design: The data block shall be 64-QAM demodulated based on the constellation shown in Chapter II. The described function can be similarly illustrated as in Figure 32.

h. Remove Guard Subcarriers

Component name: *guardRem_802_16*

Port design: *guardRem_802_16* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: The data block shall have null data, equivalently subcarriers, removed from its front and end as described in Chapter II. The described function is illustrated in Figure 33.

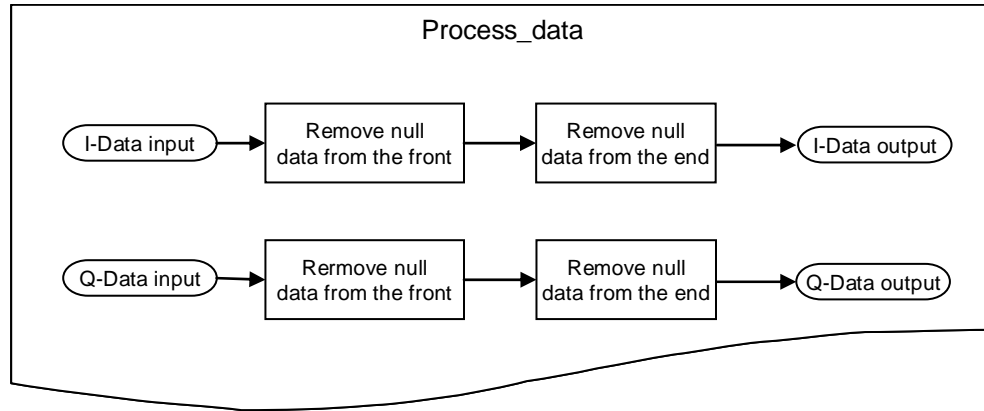


Figure 33. Functional description of component *guardRem_802_16*.

i. Remove Pilot Tone and DC Null Subcarriers

Component name: *ptRem_802_16*

Port design: *ptRem_802_16* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: The data block shall have pilot tone and DC null data, equivalently subcarriers, removed from prescribed locations of the incoming data block as described in Chapter II. The described function is illustrated in Figure 34.

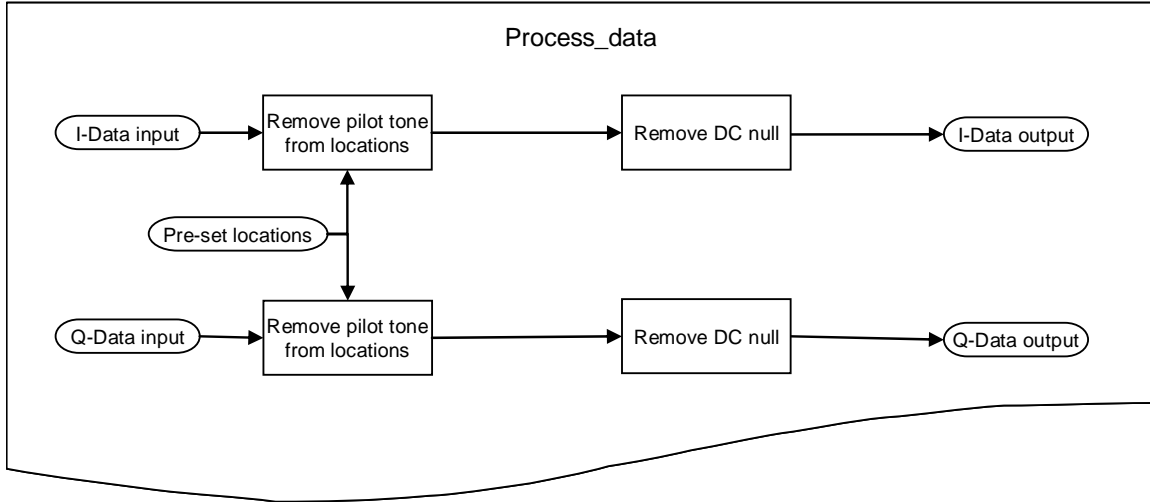


Figure 34. Functional description of component *ptRem_802_16*.

j. Fast Fourier Transform (IFFT)

Component name: *Data_IFFT*

Port design: *Data_IFFT* has one input and one output port. Both are of type *ComplexFloat*.

Functional design: It implements the Fast Fourier Transform for the OFDM demodulation. The input data are taken as time samples and converted to frequency samples using the Decimation-In-Time (DIT) Permutated Input - Natural Output (PINO) FFT algorithm as described by Leong in [22]. The described function is illustrated in Figure 35. [22]

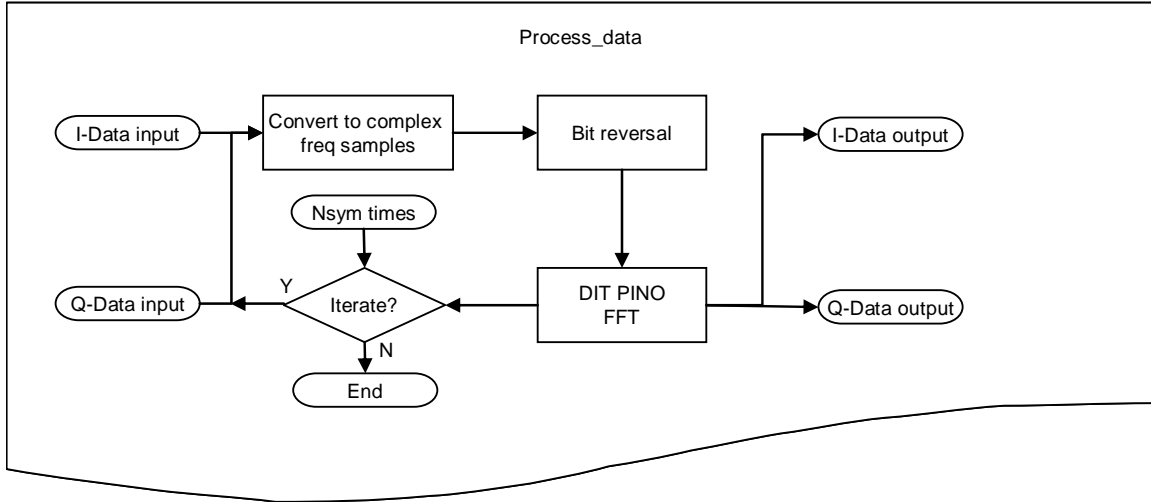


Figure 35. Functional description of component *Data_FFT* (After [22]).

k. Remove Cyclic Prefix

Component name: *cpRem_802_16*

Port design: *cpRem_802_16* has one input and output port. Both are of type *ComplexFloat*.

Functional design: The data block shall have the cyclic prefix, equivalently a block of duplicated subcarriers, removed from its front as described in Chapter II. The proportion of the data to duplicate and append is to be pre-set. The described function is illustrated in Figure 36.

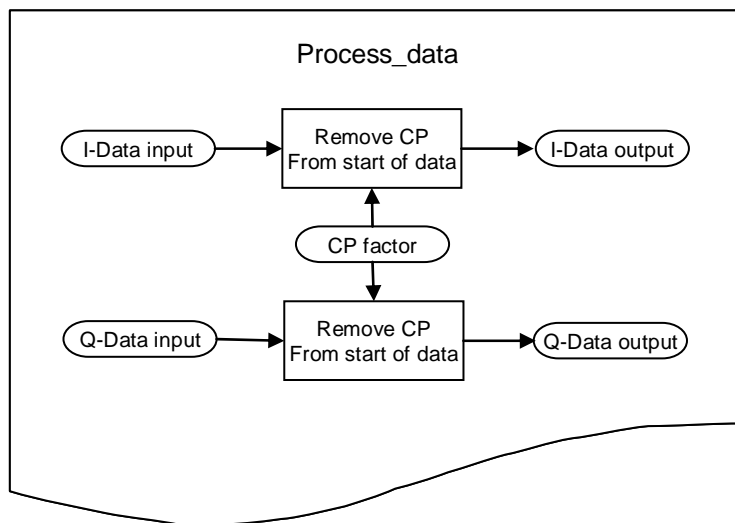


Figure 36. Functional description of component *cpRem_802_16*.

E. SUMMARY

This chapter presented the process of software development. The Incremental Development Model was adopted as the approach to develop the software. The considerations taken for the software development were discussed. The design of the waveform and components were also explained. The next chapter will present the software tests and results. For the former, the test methodology will be described. For the latter, the results with respect to meeting the objectives of the thesis will be presented.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SOFTWARE TESTS AND RESULTS

A. TEST METHODOLOGY

A convenient way of testing the developed software will be to verify the functionality of the waveform by having it receive an actual external transmission of an IEEE 802.16-2004 standard signal in the air. However, it was not feasible as there is no actual hardware interface implementation carried out in this thesis. The waveform was also not complete given that the Reed-Solomon encoder and decoder were not developed. In addition, the waveform design for the receiver does not provision for frequency or phase synchronization, symbol synchronization, fading mitigation and packet detection, which are necessary functions for a working receiver.

The overall software development approach using the Incremental Development Model remains the same where software testing is an integral part of each increment cycle. For each increment cycle, software testing was done first on individual components before subsequent testing of the incremented waveform. This methodology is elaborated in Figure 37.

A test case is available in the IEEE 802.16-2004 standard document to validate the functionality of the individual components. The details of the test case are summarized in Table 8. The SDR transmitter components were first tested individually through a simple waveform set-up shown in Figure 38. The results were then verified against the test case to validate functionality of the component.

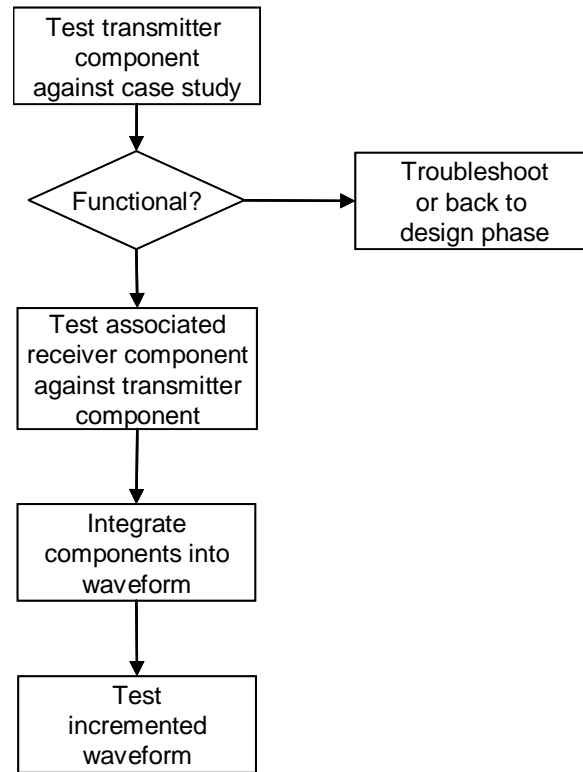


Figure 37. Software test methodology.

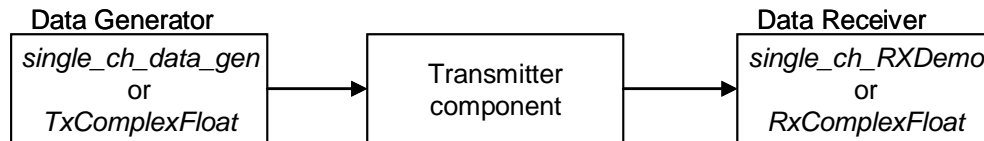


Figure 38. Set-up for testing a transmitter component.

In the simple waveform set-up for component testing, the *single_ch_data_gen* component was created to output a single-channel binary sequence. The *TxComplexFloat* component was created to output dual-channel data in real-numbered format. The *single_ch_RXDemo* component was created to receive a single-channel binary sequence. The received data was then written into a text file named *single_ch_RXDemo.txt* for analysis and verification against the test case. The *RxComplexFloat* component was created to receive dual-channel data in real-numbered format. Similarly, the received data was written into a text file named *RxComplexFloat.txt*. Depending on the type of data the component under test is required to receive and transmit, the appropriate data generator and receiver would be deployed accordingly.

Having successfully tested the transmitter component and had its functionality validated against the test case, the corresponding receiver component would then be tested against this transmitter component. The functionality of the receiver component was validated by verifying that the data, ported to a text file, received by the data receiver was the same as that sent out by the data generator. An example of this set-up is shown in Figure 39.

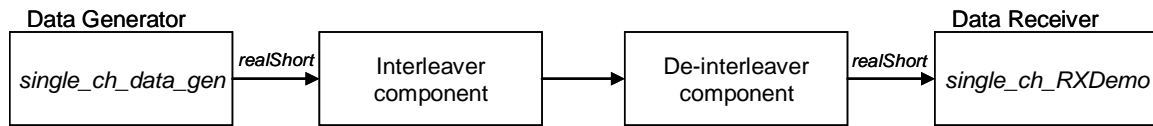


Figure 39. An example of set-up for testing a receiver component.

Following the successful tests of the transmitter and corresponding receiver components, they would then be integrated into the waveform as increments and were tested as a waveform. An example of this set-up is shown in Figure 40.

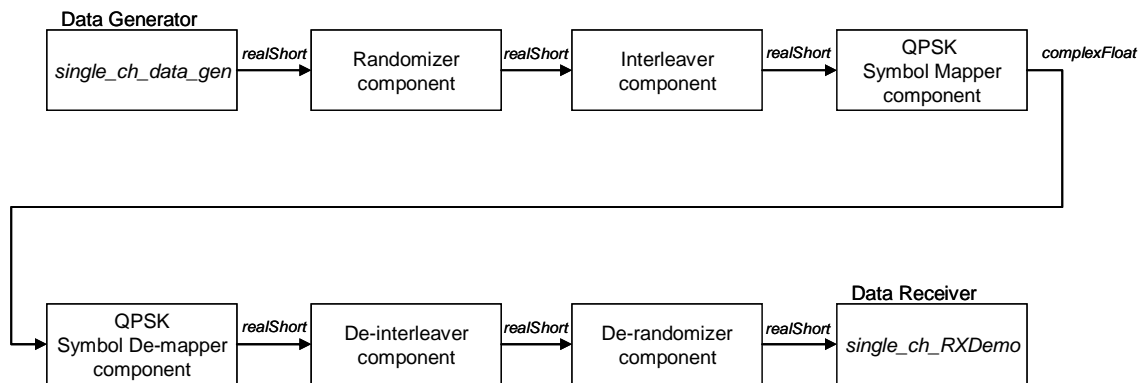


Figure 40. An example of set-up for testing an incremented waveform.

One burst of OFDM uplink data. Modulation mode: QPSK, rate 3/4	
Input Data (Hex)	45 29 C4 79 AD 0F 55 28 AD 87 B5 76 1A 9C 80 50 45 1B 9F D9 2A 88 95 EB AE B5 2E 03 4F 09 14 69 58 0A 5D
Randomized Data (Hex)	D4 BA A1 12 F2 74 96 30 27 D4 88 9C 96 E3 A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62 1A C1
Reed-Solomon Encoded Data (Hex)	49 31 40 BF D4 BA A1 12 F2 74 96 30 27 D4 88 9C 96 E3 A9 52 B3 15 AB FD 92 53 07 32 C0 62 48 F0 19 22 E0 91 62 1A C1 00
Convolutionally Encoded Data (Hex)	3A 5E E7 AE 49 9E 6F 1C 6F C1 28 BC BD AB 57 CD BC CD E3 A7 92 CA 92 C2 4D BC 8D 78 32 FB BF DF 23 ED 8A 94 16 27 A5 65 CF 7D 16 7A 45 B8 09 CC
Interleaved Data (Hex)	77 FA 4F 17 4E 3E E6 70 E8 CD 3F 76 90 C4 2C DB F9 B7 FB 43 6C F1 9A BD ED 0A 1C D8 1B EC 9B 30 15 BA DA 31 F5 50 49 7D 56 ED B4 88 CC 72 FC 5C
Interleaved Data (Hex)	-100: 1 -1, -99: -1 -1, -98: 1 -1, -97: -1 -1, -96: -1 -1, -95: -1 -1, -94: -1 1, -93: -1 1, -92: 1 -1, -91: 1 1, -90: -1 -1, -89: -1 -1, -88:pilot= 1 0, -87: 1 1, -86: 1 -1, -85: 1 -1, -84: -1 -1, -83: 1 -1, -82: 1 1, -81: -1 -1, -80: -1 1, -79: 1 1, -78: -1 -1, -77: -1 -1, -76: -1 1, -75: -1 -1, -74: -1 1, -73: 1 -1, -72: -1 1, -71: 1 -1, -70: -1 -1, -69: 1 1, -68: 1 1, -67: -1 -1, -66: -1 1, -65: -1 1, -64: 1 1, -63:pilot= -1 0, -62: -1 -1, - 61: 1 1, -60: -1 -1, -59: 1 -1, -58: 1 1, -57: -1 -1, -56: -1 -1, -55: -1 -1, -54: 1 -1, -53: -1 -1, -52: 1 -1, -51: -1 1, -50: -1 1, -49: 1 -1, -48: 1 1, -47: 1 1, -46: -1 -1, -45: 1 1, -44: 1 -1, -43: 1 1, -42: 1 1, -41: -1 1, -40: -1 -1, -39: 1 1, -38:pilot= 1 0, -37: -1 -1, -36: 1 -1, -35: -1 1, -34: -1 -1, -33: -1 -1, -32: -1 -1, -31: -1 1, -30: 1 -1, -29: -1 1, -28: -1 -1, -27: 1 -1, -26: -1 -1, -25: -1 -1, -24: -1 -1, -23: -1 1, -22: -1 -1, -21: 1 -1, -20: 1 1, -19: 1 -1, -18: -1 -1, -17: 1 -1, -16: -1 1, -15: -1 -1, -14: 1 1, -13:pilot= - 1 0, -12: -1 -1, -11: -1 -1, -10: 1 1, -9: 1 -1, -8: -1 1, -7: 1 -1, -6: -1 1, -5: -1 1, -4: -1 1, -3: -1 -1, -2: - 1 -1, -1: 1 -1, 0: 0 0, 1: -1 -1, 2: -1 1, 3: -1 -1, 4: 1 -1, 5: 1 1, 6: 1 1, 7: -1 1, 8: -1 1, 9: 1 1, 10: 1 -1, 11: -1 -1, 12: 1 1, 13:pilot= 1 0, 14: -1 -1, 15: 1 -1, 16: -1 1, 17: 1 1, 18: 1 1, 19: 1 -1, 20: -1 1, 21: -1 -1, 22: -1 -1, 23: -1 1, 24: -1 -1, 25: 1 1, 26: -1 1, 27: 1 -1, 28: -1 1, 29: -1 -1, 30: 1 1, 31: -1 -1, 32: 1 1, 33: 1 1, 34: 1 1, 35: 1 -1, 36: 1 -1, 37: 1 -1, 38:pilot= 1 0, 39: -1 1, 40: -1 -1, 41: -1 1, 42: -1 1, 43: -1 -1, 44: 1 -1, 45: -1 1, 46: -1 1, 47: 1 1, 48: -1 -1, 49: 1 1, 50: 1 -1, 51: -1 -1, 52: -1 -1, 53: 1 -1, 54: 1 -1, 55: 1 -1, 56: 1 -1, 57: 1 1, 58: 1 1, 59: 1 -1, 60: 1 1, 61: -1 1, 62: 1 -1, 63:pilot= 1 0, 64: 1 -1, 65: -1 -1, 66: -1 -1, 67: 1 -1, 68: 1 -1, 69: 1 -1, 70: 1 -1, 71: -1 1, 72: -1 -1, 73: -1 1, 74: -1 -1, 75: 1 -1, 76: -1 1, 77: -1 -1, 78: 1 -1, 79: 1 1, 80: -1 1, 81: 1 1, 82: -1 1, 83: 1 1, 84: -1 -1, 85: 1 1, 86: -1 -1, 87: 1 1, 88:pilot= 1 0, 89: 1 -1, 90: -1 -1, 91: 1 1, 92: -1 1, 93: -1 -1, 94: -1 -1, 95: -1 -1, 96: 1 1, 97: 1 -1, 98: 1 -1, 99: -1 -1, 100: 1 1

* Note that the above QPSK values (all values with exception of the BPSK pilots) are to be normalized with a factor $1/\sqrt{2}$

Table 8. An example of an IEEE 802.16 test case (From [10]).

B. RESULTS

This section presents the results of the thesis work with respect to the goals of the thesis which can be categorized into two aspects of the developed software functionality and reusability.

1. Functionality

The test methodology described above in Section A serves to validate the functionality of the individual components. All transmitter and receiver components required to build the waveform compliant to the physical layer of the IEEE 802.16 WirelessMAN-OFDMTM, except the Reed-Solomon encoder and decoder, were tested and had their functionality validated successfully.

2. Reusability and Reconfigurability

Reusability is an important trait of the software since the developed components are to be contributed to a shared library for future use. Closely related to reusability of components is the trait of reconfigurability of waveforms. These considerations for the software design have been elaborated upon in Chapter IV. The SDR components developed in this thesis were designed with a strong emphasis on reusability and reconfigurability. The following describes the features incorporated into the design of the components that strive to achieve these traits.

a. Documentation

Proper commentary was inserted in all the source code for the software. The commentary shall serve to explain in further detail the workings of the software algorithm. It also differentiates portions of the code which were manually modified or inserted from those that were automatically generated by the OWD. These commentaries will be useful as reference for future modifications and adaptations.

In addition, for every SDR component, a description file was created that details the general parameters relevant to understanding the structural design of the component. This information will be useful as a help reference to assist in deploying the component in the OWD for the building of any waveform. In other words, no foreknowledge of the design of the components in the library is necessary to still be able to select and deploy appropriately the library components in order to build any waveform effectively. The description file is a text file stored in the component folder in the OS file system. An example of a description file is shown in Figure 41.

```

Desc - WordPad
File Edit View Insert Format Help

a. Component
- BPSK_mod_802_16

b. Input Port interface
- realShort (I integer)

c. Output Port interface
- complexFloat (I & Q float)

d. Function
- Receives single serial stream of I channel representing logic data and BPSK modulates
it iaw IEEE802.16 before sending a reduced packet size of modulated I & Q channels out.

e. Packet size (iaw IEEE802.16 std)
- Input: dynamic - typically 192 in a OFDM symbol
- Output: input packet size

f. Programmed by : John Low - 17th Aug 2006

g. OSSIE version: 0.5.0

IEEE802.16 (WirelessMAN-OFDM)
-----
BPSK shall be supported. The constellations shall be normalized by multiplying the
constellation point with the indicated normalising factor (1.0) to achieve equal average
power.

Constellation as shown below:

-----
Bit |      B0      |
-----
   | 0 | 1 |
-----
I  | 1 | -1 |
-----
Q  | x | x |
-----

B0 denotes LSB.
The first bit out of the interleaver shall map to the MSB in the constellation.

For Help, press F1

```

Figure 41. An example of a SDR component description file.

b. Naming Convention

As much as it is desired for the SDR components to be generic and elementary such that they are applicable for building all kinds of waveform profiles, fundamental attribute differences between waveform requirements may dictate otherwise. For example, the constellation for the IEEE 802.16-2004 standard is different from that specified in the IEEE 802.11a standard. This dictates that there cannot be a generic symbol mapping component that can satisfy the requirements of both standards without having to change some parameters which means having to recompile the component each

time it is used differently. This inconvenience of having to recompile the component may be circumvented by having a control port interface, in addition to the standard data port interface, which detects this parameter change. However, this structural design of the component may inadvertently complicate the whole waveform design since it is expected that the standard communication signal does not cater for such a parameter change i.e., provision for differentiation of the communications standard. The control signal that triggers the parameter change within the component will then have to be internally generated within the waveform thus complicating the design.

It would be more feasible to allow in the library, where applicable, multiple components of the same function but different attributes. For example, there can be a BPSK symbol mapper compliant to the IEEE 802.16-2004 standard and another compliant to the IEEE 802.11a. This will alleviate the need to change parameters and recompile the component as well as maintain the components in their elementary functional structure. Therefore, the naming convention of the components is important to differentiate the components accordingly in the library. For example, the BPSK symbol mapper compliant to the IEEE 802.16-2004 standard has been named *bpsk_mod_802_16* whereas another compliant to the IEEE 802.11a may be named *bpsk_mod_802_11a*.

c. Dynamic Data Size

In OSSIE, the data flow between components is in terms of packets. The size of the packet varies according to the waveform signal profile. For example, in the IEEE 802.16 WirelessMAN-OFDMTM standard, data flow is in terms of an OFDM symbol which comprises of 256 subcarriers. For the IEEE 802.11a standard, an OFDM signal comprises only of 64 subcarriers.

As described above, having to recompile a component due to parameter change to suit different waveform profile requirements does not align well with the trait of component reusability. Therefore, it is important that the components be designed with a dynamic ability to receive and handle different data sizes.

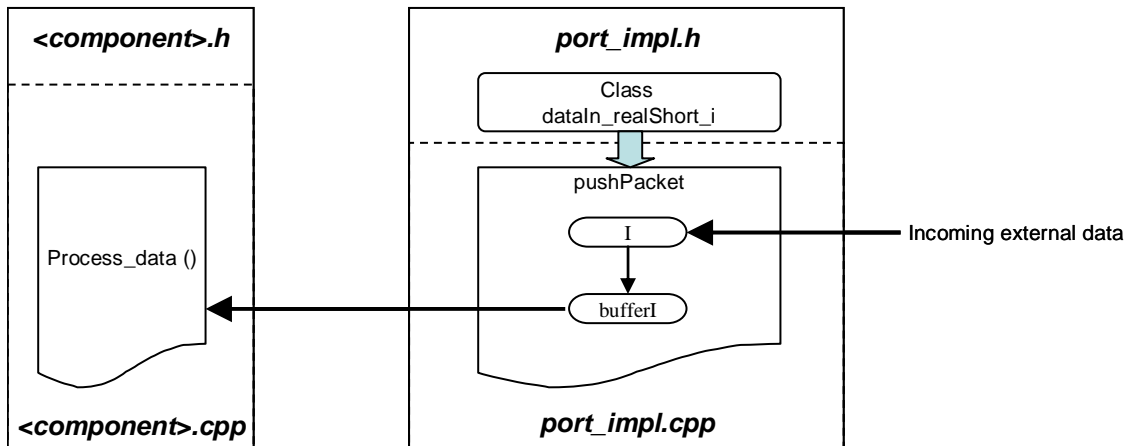


Figure 42. An example of data handling within a component.

Figure 42 above shows how OSSIE sets up the component structural design to handle an incoming data packet. The data packet is first received by the function *pushpacket* which accesses the data from a common buffer used for data transfer throughout the waveform. Once fully accessed, this data is then copied onto an interim buffer for data processing within the component. As such, the size of the data to be handled by the component can be ascertained during the operation within *pushpacket*, and initialized for subsequent processing by *process_data*.

d. Elementary Functional Design

As described in Chapter IV, the SDR components were designed to have a single elementary function. This feature enables the components to serve as basic building blocks in building any waveform. For example, instead of developing a symbol mapper component that encompasses functions of BPSK, QPSK, 16-QAM and 64-QAM, four separate components were developed that each embodies a single function. In the former design, the inconveniences that will be incurred in using the component were described above under the section on the importance of the component naming convention. The latter design will enhance reusability of the components and reconfigurability of the waveform. This design will also be very useful for waveforms that support multi-mode operations which will be elaborated in greater detail below in Chapter VI.

e. Generic Port Interface Structure

The component design restricted the port interfaces of each component to be kept simple and elementary and only data port interfaces were used. Having additional control data interfaces will incur complexity in waveform design as described above under the section on the importance of the component naming convention. Keeping port interfaces simple and generic is important in ensuring reusability of the component and reconfigurability of the waveform. This can be illustrated in a waveform set up for multi-mode operation which is elaborated in greater detail below in Chapter VI.

C. SUMMARY

This chapter presented the software test methodology and the results with respect to meeting the thesis objectives. All the transmitter and receiver components required to build the waveform compliant to the physical layer of the IEEE 802.16 WirelessMAN-OFDM™, except the Reed-Solomon encoder and decoder, were tested and validated successfully. Several features were incorporated into the component design to good reusability of the components. These features include having good documentation, a useful component naming convention, dynamic data size handling capability, a simple functional design and generic port interfaces. The next chapter will present additional design work which focuses on a suggested architecture that allows multi-mode operations without compromising the reusability of the components. An experiment to design and test the architecture will be presented and results explained.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. ADDITIONAL WORK

Although developing a fully operational receiver based on the IEEE 802.16-2004 standard was not attempted in this thesis, the work does include a suggested architecture to achieve multi-mode operations. In this chapter, the suggested architecture is described on. An experiment to design and test the architecture will be presented and results explained.

In today's data communication standards, it is common to provision for multi-mode operations. This is especially so for wireless communication where different modes have to be catered for in meeting dynamic channel characteristics. For example, in the IEEE 802.16 WirelessMan-OFDMTM standard, twenty different burst profiles are provisioned for [10]. These profiles consist of different combinations of modulation type and error correction coding scheme and rate. Control information in the header within a frame tells the receiver which burst profile to adopt to receive the following data bursts within the frame.

A waveform design that will meet such provision requirements can be complicated. A convenient way is to custom-build components that will take on different functions given a parameter change. However, components built in such a way will not be very reusable since it is customized for a particular type of waveform.

A. MULTI-MODE OPERATIONS WAVEFORM DESIGN

In this thesis, a waveform design was experimented with to cater to multi-mode operations despite using simple and reusable components. This design is shown in Figure 43. Component 1a, 1b and 1c can be separate components of the same function but of different attributes. For example, Component 1a may be a BPSK symbol mapper while Component 1b may be a QPSK symbol mapper.

In OSSIE, a component designated as an Assembly Controller will trigger the start of the waveform operations. The Selector component, designated as the Assembly Controller, will first trigger the Data Generator to send the first or the next data packet, regardless of size since all the components shall be designed with dynamic data size

handling ability. Upon receiving the data packet from the Data Generator, the Selector shall proceed to check the control information received from the Receiver. In the case the data received from the Data Generator is the first, no control information will be available from the Receiver. This also indicates that this data packet contains the header information required to select which profile to adopt to receive subsequent data packets. Meanwhile, the Selector will output the first data packet, containing the header information, to a default profile by sending it to the appropriate output interface port. This data packet will then be processed accordingly until the relevant header information is being extracted by the Receiver and then relayed back to the Selector through the control port interface. The whole cycle then repeats until the end of data transmission.

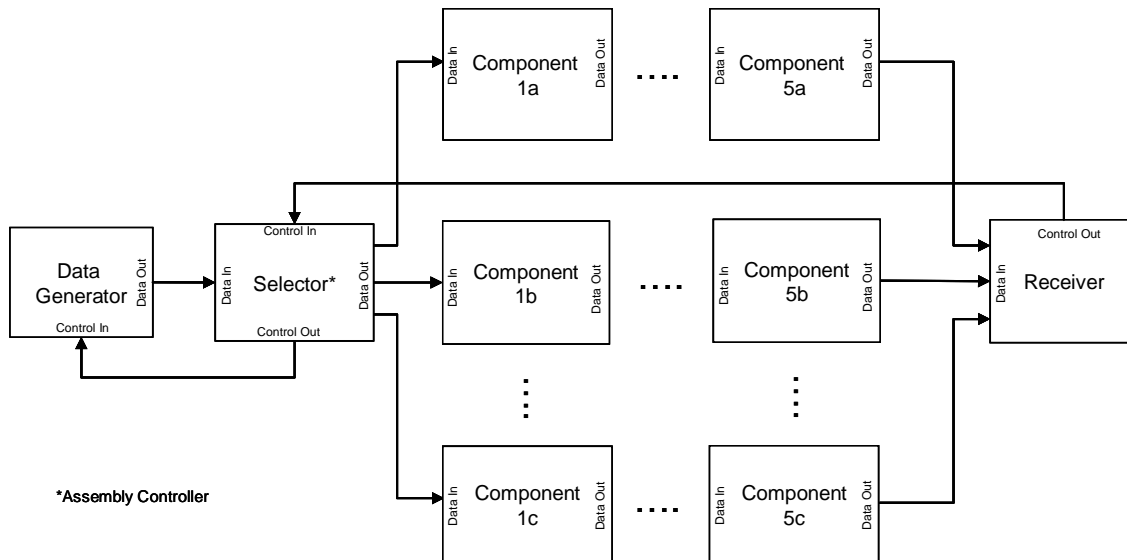


Figure 43. Conceptual waveform design for multi-mode operations.

A working model shown in Figure 44 was constructed and tested. The components Profile 1, 2 and 3 were created to represent waveforms of different profiles. In real implementation, components Profile 1, 2 and 3 could be a series of simple and reusable components.

B. MULTI-MODE OPERATIONS COMPONENT DESIGN

The key components in this waveform set-up for experimentation are the Selector and Receiver. The port interface structure of the individual components are illustrated in Figure 44. All port interfaces are of type *realShort*.

Components Profile 1, 2 and 3 were created as dummies that did no actual processing work since it would not have contributed to the goal of the experimentation. The only significant role of these components was to flag their respective profile name when data was passing through them. This was so the data flow through the waveform could be tracked and verified if the correct profile was selected.

The Data Generator, named *single_ch_data_gen_FB*, serves to output a packet of the binary data stream only upon receiving the control signal from the Selector.

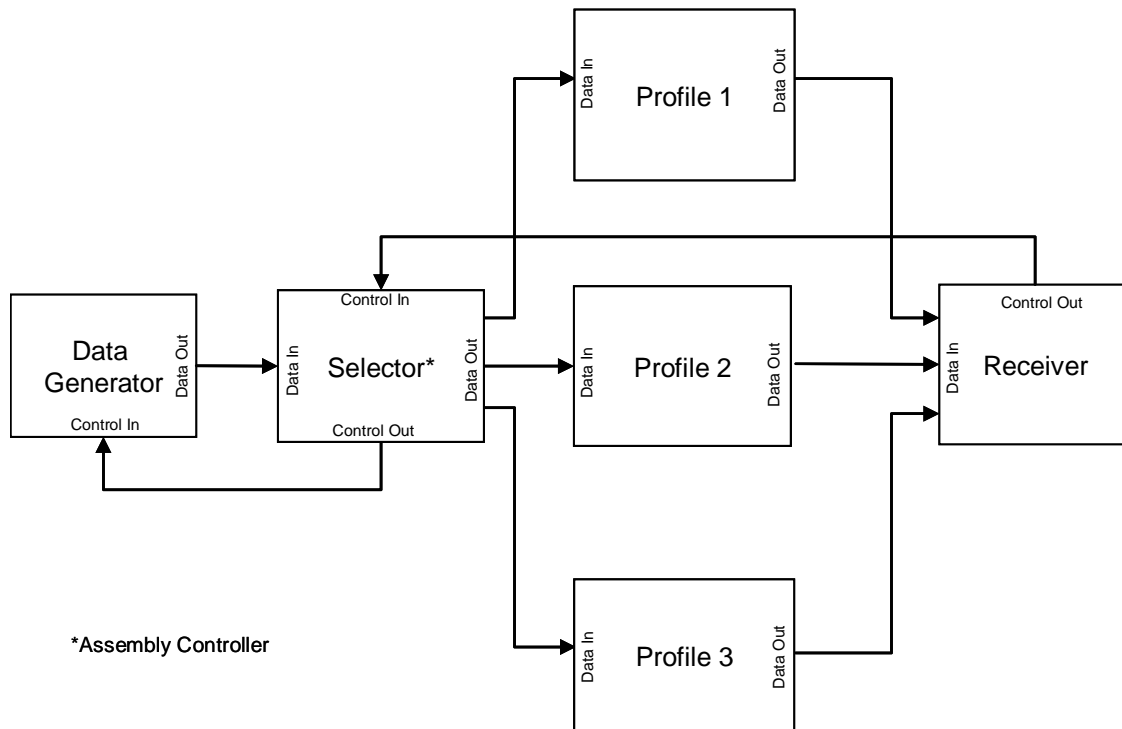


Figure 44. Working model of a multi-mode operations waveform.

The Selector, named *SelectorReal_3Out*, is designed as an Assembly Controller which means it possesses a function to initiate waveform activation. In this case, it does

so by sending a control signal to trigger the Data Generator's function. The Selector then waits for the data input from the Data Generator. Upon receiving the data packet, it then checks its input control information received from the Receiver. This information is necessary to help the Selector decide to which data port interface to channel the data packet. If there is no control signal available yet, the Selector will then proceed to channel the data packet to a default output port interface, corresponding to the lowest data rate mode, i.e. BPSK modulation with a rate 1/2 convolutional code. The functional description of the Selector is illustrated in Figure 45.

It should be noted that although OSSIE 0.5.0 allows multiple input port interface of the same type, they share a common buffer as explained in Chapter IV. In this case, the control and data input port interfaces of the Selector are of the same port type *realShort*. It was thus important in the design of the component that data stored in this common buffer are read before being overwritten by another.

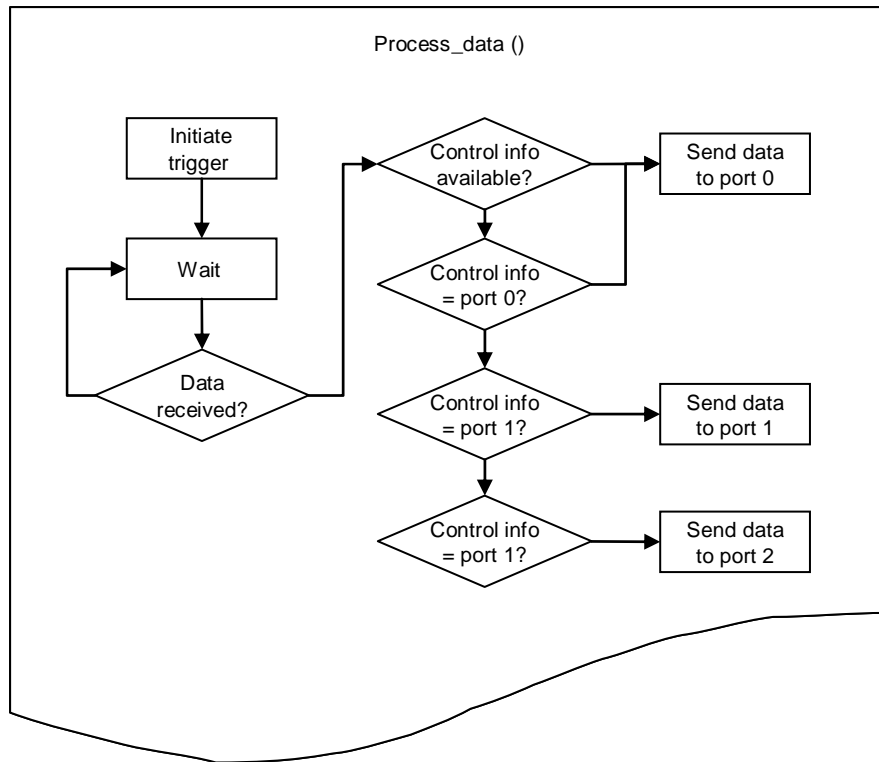


Figure 45. Functional Description of *SelectorReal_3Out*.

The Receiver's main function is to receive a data packet and extract the header information whose location on the data stream is pre-known. It then feeds this header information back to the Selector.

C. TESTS AND RESULTS

In this experimentation, the main purpose is to test the synchronization of the signal flow among the components which is critical to rendering the design feasible for multi-mode operations.

The waveform was constructed in OWD based on the working model shown in Figure 44. Data received at each component was displayed for analysis and verification that the signal flow was according to intentions. A script of the test results is shown in Figure 46. It clearly shows that the waveform was performing as intended.

This experiment demonstrated that multi-mode operations within a waveform is feasible without compromising the reusability of the components. The waveform can be reconfigured with different profiles easily.

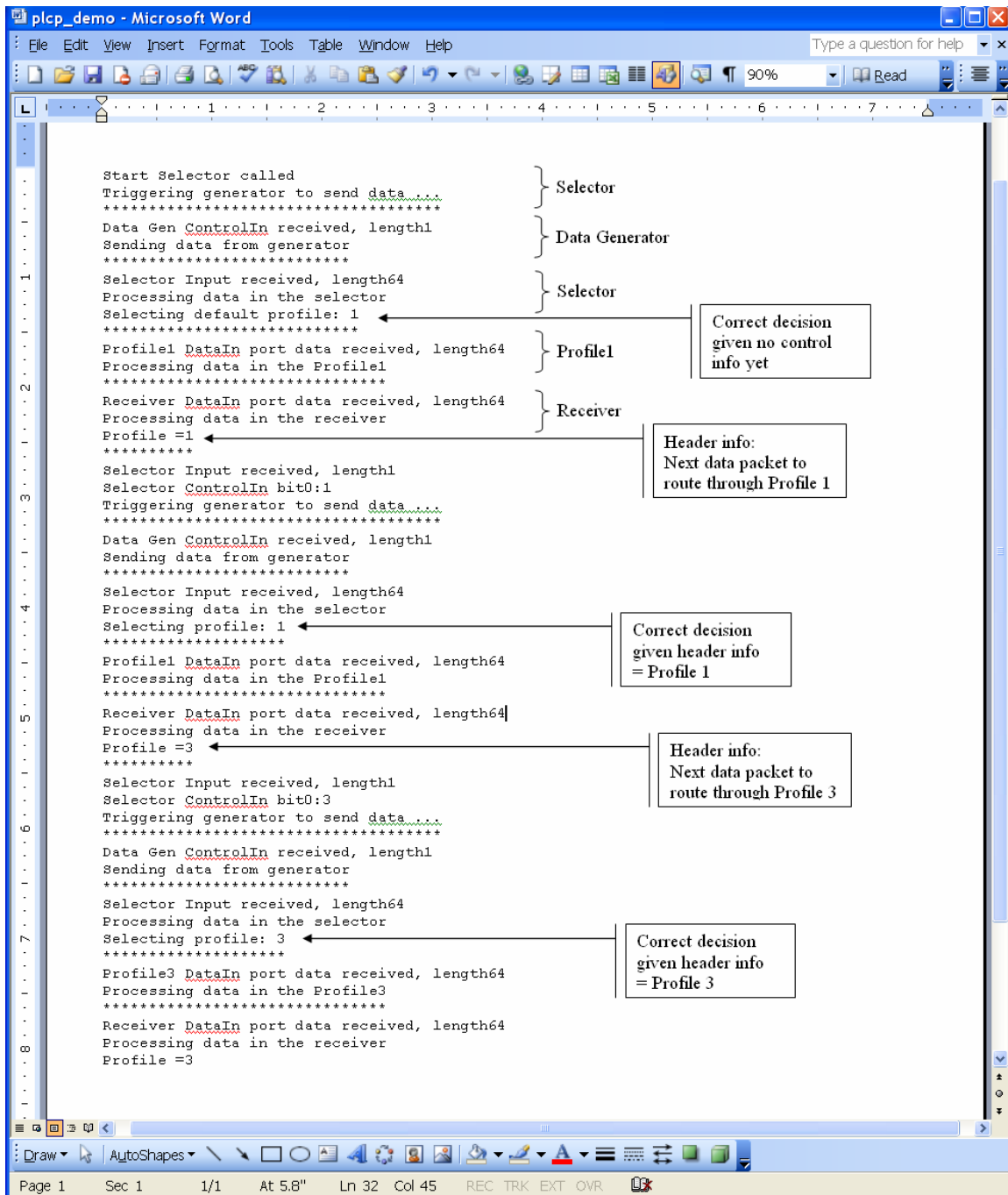


Figure 46. Script of test results for a multi-mode operations capable waveform.

D. SUMMARY

This chapter presented the additional thesis design work which focused on a suggested architecture that allows multi-mode operations without compromising the reusability of the components. An experiment to design and test the architecture was

presented and results shown which demonstrated that the suggested architecture was able to achieve the intended goal. The next chapter will present the final thesis conclusions and recommendations for future works.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The objectives of this thesis were two-fold. The first was to design and implement, using OSSIE, software defined radio transmitter and receiver components based on the IEEE 802.16 WirelessMAN-OFDMTM standard. The second objective was to ensure the developed components, to be contributed to the library, will be flexible and useful for building other transceivers .

In this thesis, all the components specified in the physical layer of the IEEE 802.16 WirelessMAN-OFDMTM standard were developed successfully except the Reed-Solomon encoder and decoder. In addition to the validated functionality, the components were developed with good reusability that will also serve to enhance reconfigurability of waveforms.

Additional work in this thesis includes successful experimentation with a suggested architectural design that accommodates simple and elementary components in a reconfigurable waveform that supports multi-mode operations.

B. RECOMMENDATIONS FOR FUTURE WORK

The software components developed in this thesis and the successful results of experimentation with a multi-mode operations waveform design serve as a baseline to implement a fully functional IEEE 802.16 WirelessMAN-OFDMTM transceiver. Additional software components to be developed include the Reed-Solomon encoder and decoder, the waveform profile selector and the receiver at the end of the waveform as well as components to handle frequency and phase synchronization, packet detection, ranging, power control, symbol synchronization and mitigation of fading [10]. Components are needed also to interface the waveform with the RF front-end implemented in hardware.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] SDR forum, "What is Software Defined Radio?", <http://www.sdrforum.org/pages/aboutTheForum/faqs.asp>, Retrieved October 2006.
- [2] J.H. Reed, "Software Radio: A Modern Approach to Radio Engineering", 1st edition New Jersey: Prentice Hall, 2002.
- [3] P.G. Cook and W. Bonser, "Architectural Overview of the SPEAKeasy System", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 4, April 1999.
- [4] Space and Naval Warfare Systems Command, "JTRS Overview", <http://enterprise.spawar.navy.mil/body.cfm?type=c&category=27&subcat=60>, Retrieved October 2006.
- [5] WIKIPEDIA The Free Encyclopedia, "Software-defined radio", [http://en.wikipedia.org/wiki/Software-defined_radio#Joint Tactical Radio System_28JTRS.29](http://en.wikipedia.org/wiki/Software-defined_radio#Joint_Tactical_Radio_System_28JTRS.29), Retrieved October 2006.
- [6] WIKIPEDIA The Free Encyclopedia, "Joint Tactical Radio System", [http://en.wikipedia.org/wiki/Joint Tactical Radio System](http://en.wikipedia.org/wiki/Joint_Tactical_Radio_System), Retrieved October 2006.
- [7] Software Defined Radio (SDR) with OSSIE Open Source SCA, <http://ossie.mprg.org/>, Retrieved October 2006.
- [8] Joseph Mitola, III, "Software Radio Architecture: Object Oriented Approaches to Wireless Systems Engineering", John Wiley and Sons, 2000.
- [9] Joseph Mitola, "The Software Radio Architecture." *IEEE Communications Magazine*, vol 33, issue 5, May 1995, pg 26-27.
- [10] LAN/MAN Standards Committee of the IEEE Computer Society and the IEEE Microwave Theory and Techniques Society, "IEEE Std 802.16TM-2004, IEEE Standard for Local and metropolitan area networks - Part 16: Air Interface for Fixed Broadband Wireless Access Systems", October 2004.
- [11] Todor Cooklev, "Wireless Communication Standards: A Study of IEEE 802.11TM, 802.15TM, and 802.16TM", chapter 4, IEEE Press Publications, August 2004.
- [12] The IEEE 802.16 Working Group on Broadband Wireless Access Standards, "Background Information on IEEE 802.16", <http://grouper.ieee.org/groups/802/16/pub/background.html>, Retrieved October 2006.
- [13] WIKIPEDIA The Free Encyclopedia, "IEEE 802.16", http://en.wikipedia.org/wiki/IEEE_802.16, Retrieved October 2006.

- [14] G.Abowd, R.Allen and D.Garlan, "Using Style to Understand Descriptions of Software Architecture", Proceedings of SIGSOFT'93: Symposium on the Foundation of Software Engineering, December 1993.
- [15] JTRS Overview website, <http://www.jtrs.saalt.army.mil/overview>, Retrieved October 2006.
- [16] Modular Software-programmable Radio Consortium, "Software Communications Architecture Specification", MSRC-5000SCA, version 2.2, November 2001.
- [17] Joint Tactical Radio System (JTRS) Joint Program Office, "Software Communications Architecture Specifications", 2nd edition, April 2004.
- [18] Sabri Murat Biçer, "A Software Communications Architecture Compliant Software Defined Radio Implementation", Master's Thesis, Northeastern University, June 2002.
- [19] WIKIPEDIA The Free Encyclopedia, "Common Object Request Broker Architecture", http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture, Retrieved October 2006.
- [20] Jacob A. DePriest, "A Practical Approach to Rapid Prototyping of SCA Waveforms", Master's Thesis, Virginia Polytechnic Institute and State University, April 2006.
- [21] WIKIPEDIA The Free Encyclopedia, "Iterative and incremental development", http://en.wikipedia.org/wiki/Iterative_and_incremental_development, Retrieved October 2006.
- [22] Leong Wai Kiat, "Software Communications Architecture (SCA) Compliant Software Defined Radio Design for IEEE 802.11a Transceiver", Master's Thesis, Naval Postgraduate School, December 2006.
- [23] Andrew S. Tanenbaum, "Modern Operating Systems", 2nd edition, New Jersey: Prentice Hall, 2001.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California
3. Professor Jeffrey Knorr, Chairman, Code EC
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California
4. Assistant Professor Frank Kragh
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA
5. Professor Clark Robertson
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA
6. Professor Charles W.Bostian
Virginia Polytechnic Institute and State University
Blacksburg, VA
7. Professor Jeffrey H. Reed
Virginia Polytechnic Institute and State University
Blacksburg, VA
8. Professor Carl Dietrich
Virginia Polytechnic Institute and State University
Blacksburg, VA
9. Mr Nathan Beltz
SPAWAR Systems Center
San Diego, CA
10. Mr Howard Pace
JTRS Joint Program Executive Office
San Diego, CA

11. Dr Richard North
JTRS Joint Program Executive Office
San Diego, CA
12. Ms Donna Miller
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA
13. MAJ Low Kian Wai
Ministry of Defence
Singapore